Extracción de información en informes médicos

Proyecto de Sistemas Informáticos

Facultad de Informática

Universidad Complutense de Madrid

Departamento de Ingeniería del Software e Inteligencia Artificial

Curso 2012/2013

Lucía Hervás Martín

Víctor Martínez Simón

Irene Sánchez Martínez

Director:

Alberto Díaz Esteban

Lucía Hervás Martín, Víctor Martínez Simón e Irene Sánchez Martínez autores del presente documento y del proyecto "*Extracción de información en informes médicos en inglés*" autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

21 de Junio de 2013

Lucía Hervás Martín Víctor Martínez Simón Irene Sánchez Martínez

Este trabajo no sería posible si no hubiésemos contado con el apoyo de las personas que nos rodean. Así, queremos dedicarlo en primer lugar a nuestro director de proyecto Alberto Díaz Esteban, por su interés y colaboración. Además a cada uno de los profesores que nos han guiado durante estos años, sin todo lo que hemos aprendido de ellos no habríamos sido capaces de desarrollar este proyecto. Y por último y por supuesto, también a nuestras familias, sin ellas no estaríamos hoy aquí, y a nuestros amigos, compañeros y parejas por todo su apoyo y por habernos soportado en los momentos de desánimo.

Resumen

El acceso a la información contenida dentro de un informe médico es vital tanto para la investigación como para el tratamiento de los pacientes. Sin embargo, la información relevante suele estar escrita en lenguaje natural, por lo su procesamiento automático no es una tarea trivial. Con este objetivo en mente, hemos desarrollado un sistema capaz de obtener un archivo que represente el contenido más relevante de un informe clínico. Como parte de esta representación se deberán detectar aquellos conceptos médicos pertenecientes a una de las ontologías más utilizadas en este ámbito, *UMLS*. Además previamente se realizará un proceso automático de corrección ortográfica, expansión de acrónimos y detección de frases afirmadas, negadas y especuladas. Todo esto en dos de los idiomas más hablados a nivel mundial: español e inglés. Esta representación permitirá a su vez desarrollar aplicaciones que la utilicen, por lo que se ha implementado también un buscador de informes médicos como ejemplo de ello.

Por último, como parte de este trabajo, también se incluye todo el proceso seguido durante nuestra participación en el *Conference and Labs of the Evaluation Forum* del año 2013, una de las organizaciones más conocidas a nivel internacional en el campo de la recuperación de información, así como el artículo científico desarrollado para la misma.

Palabras clave: Procesamiento del lenguaje natural, recuperación de información, informe médico, detección de conceptos, buscador, *UMLS*, MetaMap, *CLEF 2013*.

Abstract

The information inside a medial report it's very important for researchers and for the patient. But this information is usually written in natural language, so automatic processing isn't a trivial task. With this target in mind, we developed a system that is able to generate a representation which contains the most relevant information in a medical report. It detects medical concepts from one of most popular biomedical ontologies, UMLS. Previously will also perform a spelling correction, acronym expansion and affirmed, negated and speculated sentences detection. All this process could be executed into the two most spoken languages in the world, English and Spanish. The representation will allow us to develop applications that use it. In fact it's been including a searcher for medical reports to show an example of what can be done with our software.

Finally, as part of this work, we explain our experience in our participation into the Conference and Labs of the Evaluation Forum 2013, a self-organized body whose is well-known in the international IR community, and the paper generate for it.

Keywords: Natural language processing, information retrieval, medical report, concepts detection, searcher, *UMLS*, MetaMap, *CLEF 2013*.

Índice general

Índice de tablas

Índice de figuras

Índice de listados de código

Capítulo 1

Introducción

1.1. Motivación de la propuesta

Con la aparición de la informática y especialmente de Internet, la explosión de información producida ha ido creciendo a pasos agigantados. Estamos situados en una época en la que los avances han permitido generar, almacenar y distribuir gran cantidad de información, en su mayor parte digitalizada, y hoy en día podemos adquirir o comunicar prácticamente cualquier tipo conocimiento desde nuestros ordenadores. Sin embargo, igualmente, cada día que pasa es más notable la dificultad que encontramos a la hora de hallar aquella que realmente nos interesa, descartando la que para nosotros resulta irrelevante. Esto se debe a una sobrecarga de información o *infoxication,* que nos afecta obligándonos a dedicar más tiempo del que nos gustaría, o, incluso en ocasiones del que podríamos disponer, a tareas relacionadas con la obtención de datos. El tiempo requerido para extraerlos es tal que comienza a resultar interesante, y finalmente necesaria, la presencia de herramientas que ayuden a los profesionales. De esta manera, la recuperación, gestión, clasificación y el tratamiento de la información han ido cobrando importancia hasta el punto de considerarse algo fundamental.

Dentro de este campo, aparece el tema que nos concierne, el del tratamiento de información biomédica y relativa a la salud. Internet resulta una fuente imprescindible en muchas ocasiones tanto para pacientes, como para profesionales pero, como hemos mencionado, estos deben enfrentarse a los efectos colaterales de la evolución de la informática; es ahí donde hace aparición nuestra propuesta.

La salud siempre ha sido tema de preocupación, por motivos evidentes, para el ser humano y uno de los mayores obstáculos relacionados con ello es el desconocimiento. Desconocimiento de la enfermedad que puede padecer un paciente o, por causas no solo burocráticas, sino también de exceso de información como mencionábamos anteriormente, de cuáles son los datos relevantes incluidos en su historial clínico. El informe médico de un individuo tiene, por tanto, suma importancia, pero esta no se limita solo a dicho individuo, es habitual que el humano se apoye en lo conocido, en la experiencia, para sacar conclusiones en situaciones similares como haría un sistema de razonamiento basado en casos. Así, no sería descabellado que un profesional de la medicina utilizase como apoyo diferentes informes médicos además del historial clínico con que trata en un determinado momento. Pero de nuevo volvemos al punto anterior: el historial clínico de una persona

puede ser muy abundante, puede abarcar gran cantidad de datos que, en función de su importancia, deberán tenerse en mayor o menor consideración. Anotaciones que contienen muchas declaraciones sobre el pasado de una persona y que pueden ser esenciales para su futuro. Siendo así, su extracción automática no es un asunto trivial existiendo numerosos estudios y conferencias[1][2] sobre este apartado.

Este proyecto, es la continuación lógica del proyecto "Procesador automático de informes médicos" elaborado durante el curso académico 2011-2012 por Enrique Bautista Barahona, Ignacio Salcedo Ramos y Alberto Ureña Herradón y dirigido por Alberto Díaz Esteban y Laura Plaza Morales [5]. Dicho procesador se encargaba de la corrección ortográfica, expansión de acrónimos, detección de las negaciones y extracción de conceptos en informes médicos en español. En el capítulo 2 se entrará más en detalle sobre el grado en que este proyecto toma como punto de partida el anterior, y las diferencias existentes entre ambos en cuanto al uso de herramientas y recursos.

1.2. Objetivos

Nuestro propósito es solventar en la mayor medida posible todos los problemas comentados en el apartado anterior, de una sola vez, mediante el proyecto sobre el que versa este trabajo. Su fin consiste en extraer aquello que pueda considerarse significativo de informes clínicos en inglés para obtener una representación suficientemente genérica de estos. Dicha representación facilitaría el desarrollo de aplicaciones, que usándola, permitiesen agilizar tareas tan notables como la búsqueda de informes médicos mediante una consulta, la recuperación de informes médicos similares a uno dado, o la obtención de datos estadísticos basados en un conjunto informes médicos. Además incluiría la información más relevante del informe, organizada por secciones en función de su contenido y descartaría aquella que no se considerase necesaria. Concretamente almacenaríamos para cada sección del informe, el conjunto de conceptos que se encontrasen en ella, acompañados no solo de su nombre e identificador, sino también de un atributo denominado estado, que podrá tener como valor "afirmado", "negado" o "especulado". Esto se debe a que no supone lo mismo decir que paciente "tiene una fractura de peroné" que decir que "tal vez tenga una fractura de peroné" ni que afirmar que "no tiene una fractura de peroné". Así, deberemos distinguir hasta tres estados para un mismo concepto, en función de la manera en que aparezca en el informe médico, todo ello con la finalidad de aportar más información significativa sobre este.

Además dichos documentos no solo podrían estar redactados en un idioma tan sumamente extendido como el inglés, sino también en español. Una de las razones decisivas que nos llevaron a escoger este lenguaje fue, además de su evidente importancia en la comunicación a todos los niveles, la cantidad de recursos que podríamos encontrar disponibles y que nos darían la posibilidad, en teoría, de conseguir una aplicación de mayor calidad. Sin embargo, dado que, tal como se ha comentado previamente, esta propuesta surge como continuación de un proyecto realizado con anterioridad y que se limitaba al uso del español, también se decidió ampliar nuestros objetivos incluyendo este último idioma y mejorando el sistema del que partíamos.

[1] http://trec.nist.gov/
[2] http://www.clef-initiative.eu//

Para concluir podemos afirmar que, en términos generales, aspiramos a utilizar los conocimientos que hemos adquirido a lo largo de estos años para poder colaborar con el ámbito médico de manera que consigamos una herramienta que pueda resultar útil y ayude a mejorar el día a día tanto de estos profesionales o de los individuos relacionados con la investigación en este campo, como, en consecuencia, de los pacientes.

1.3. Estructura del documento

El presente documento sigue una división en varios capítulos, los cuáles describimos brevemente a continuación:

- **Capítulo 1. Introducción.** Detalla cuál es la motivación de nuestra propuesta y los objetivos que perseguimos con la implementación de este proyecto. Por último anticipa la organización del documento.
- **Capítulo 2. Estado de la cuestión.** Describe y analiza el estado de la cuestión en lo relativo al procesamiento del lenguaje natural y al tratamiento de información médica y presenta los recursos y herramientas disponibles cuyo uso ha sido planteado para el desarrollo de este trabajo. Además expone y justifica cuáles han sido las decisiones tomadas a la hora de seleccionar algunas de ellas y explica cuál ha sido el punto de partida para el comienzo de dicho desarrollo.
- **Capítulo 3. Sistema de procesamiento de informes.** Especifica cuál es la funcionalidad proporcionada por el sistema haciendo mención al funcionamiento interno de cada una de las partes que lo componen.
- **Capítulo 4. Usos y aplicaciones.** A lo largo de este capítulo se tratará de explicar cuáles son las posibles aplicaciones que tiene el trabajo que hemos realizado. Además en él se incluirá una sección relativa a la edición del *CLEF 2013* y a nuestra participación en él.
- **Capítulo 5. Arquitectura del sistema.** Definirá la arquitectura del sistema implementado, centrándose en la vista lógica, con el objetivo de hacer comprender al lector y al posible desarrollador que se encuentre interesado en él, la estructura interna de este.
- **Capítulo 6. Conclusiones y trabajo futuro.** Expone los resultados alcanzados y las ampliaciones potenciales del trabajo realizado.
- **Apéndice A. Manual de usuario.** Servirá al usuario para conocer el funcionamiento de la aplicación y la forma en que deberá utilizarlo.
- **Apéndice B. Configuración.** Detalla la estructura y el contenido de los archivos de configuración y cómo deberán ser modificados en función del objetivo con que se utilice el sistema.
- **Apéndice C. Creación de una ontología personalizada.** Explica al desarrollador los pasos a seguir en la creación de una ontología propia utilizando MetamorphoSys.

- **Apéndice D. Instalación y ejecución de MetaMap y MetaMap Java API.** Describe las acciones que deberán llevarse a cabo para conseguir una instalación completa y correcta tanto de MetaMap como de MetaMap Java API, dado que dicha instalación será imprescindible para poder hacer uso del sistema que hemos desarrollado.
- **Apéndice E. UCM at CLEF eHealth 2013.** Recoge tanto el *Extended Abstract* como el *paper* desarrollados para nuestra participación en la edición del *CLEF* 2013.
- **Apéndice F. Glosario de términos.** Recoge la definición de una serie de términos que son considerados relevantes en el proyecto que nos ocupa.

Capítulo 2

Estado de la cuestión

2.1. Introducción

El propósito de este capítulo es presentar los recursos y herramientas software disponibles, relacionados con el ámbito lingüístico y biomédico, que se han planteado como opciones a tener en cuenta a la hora de obtener apoyo para la realización del proyecto, que recordemos tiene el objetivo de extraer información de informes que estén redactados tanto en inglés como en español. Además de la presente introducción, este capítulo cuenta con las siguientes secciones:

- **Ontologías y terminologías biomédicas.** En primer lugar, puesto que el objetivo final del proyecto es obtener la representación de un informe médico mediante, entre otras operaciones, la extracción de los conceptos médicos que se encuentran en él, comenzaremos con una sección dedicada a diferentes ontologías y terminologías biomédicas de uso frecuente en tareas de Procesamiento del Lenguaje Natural (PLN o *NLP* en inglés). Concretamente pondremos especial atención a *SNOMED-Clinical Terms (SNOMED-CT)* y *Unified Medical Language System (UMLS)*.
- **MetaMap.** Esta sección tratará sobre MetaMap, una herramienta que da la posibilidad identificar los conceptos del Metatesauro *UMLS* presentes en un texto. Dada su gran utilidad, hablaremos de ella en numerosas ocasionas a lo largo del capítulo.
- **Detección y desambiguación de conceptos médicos.** Enlazando con el aparato anterior, incluimos este, en el que estudiamos las formas en que podemos detectar conceptos y realizar una desambiguación de los mismos.
- **Corrección ortográfica.** Puesto que antes de identificar y extraer conceptos médicos de los informes querremos corregir las posibles faltas de ortografía que pusiesen aparecer en él, este apartado estará dedicado a diferentes proyectos o algoritmos relacionados con la corrección ortográfica.
- **Expansión y desambiguación de acrónimos.** Esta sección se centrará en algunos de los métodos que pueden utilizarse para expandir acrónimos y en la tarea de desambiguación que aparece como necesaria al enfrentarnos a ello. Este apartado no interesará en tanto en cuanto otorguemos relevancia al hecho de que puede que haya conceptos médicos “escondidos” tras un acrónimo que será preciso expandir.

- **Detección de negaciones.** Además de detectar los conceptos será importante detectar si están negados. En este sentido, no es lo mismo decir "el paciente no tiene un esguince" que afirmar "el paciente tiene un esguince". Por ello también deberemos dedicar una parte de la implementación a la detección de negaciones. Dicho esto, resulta lógico que estudiemos cuáles son las técnicas o herramientas más empleadas para ello en la actualidad.

- **Detección de especulaciones.** Nos encontramos en un caso similar al descrito en el punto anterior. Además de detectar los conceptos nos preocupará tener en cuenta cuál es su estado. Así, la frase "el paciente podría tener un esguince" no obtendrá una representación final equivalente a la frase "el paciente tiene un esguince". De esta manera estaremos recogiendo más información asociada al concepto simplemente con indicar su estado. Es por ello que incluimos este apartado, en el que hablamos sobre cómo se aborda actualmente el problema de la detección de especulaciones.

- **Indexación y realización de consultas.** A pesar de que el objetivo principal del proyecto es la consecución de un sistema que extraiga la información contenida en informes médicos, esto a su vez tiene como meta que la representación obtenida con la información extraída, sea aprovechada con el fin de desarrollar una aplicación que la utilice. Un idea inmediata consiste en la implementación de una herramienta que permita realizar búsquedas en un conjunto de informes médicos ya procesados.

- **Conferencias relacionadas.** En esta sección se presentan diferentes conferencias que por su relación con las tareas de recuperación de información, en algunos casos dentro del campo médico, han tomado un papel importante en el desarrollo de este proyecto. En primer lugar comenzaremos presentando la *Text REtrieval Conference (TREC)* para después pasar a hablar sobre el *Conference and Labs of the Evaluation Forum (CLEF)* que toma importancia en este documento por su relación con las tareas de recuperación de información y principalmente por nuestra participación en la edición de este año 2013.

- **Punto de partida y conclusiones.** Esta última sección establece el punto de partida en la implementación del proyecto que nos ocupa, comentando las herramientas y recursos que finalmente fueron escogidos y utilizados.

2.2. Ontologías y terminologías biomédicas

En el campo de la informática, la inteligencia artificial y los sistemas basados en conocimiento podemos definir una ontología como una representación formal de un conocimiento compartido que aporta un mayor nivel de información sobre un dominio determinado que un simple diccionario o conjunto de términos o definiciones. Para una definición más concreta proponemos a continuación algunas de las cuáles han sido dadas a lo largo del tiempo.

"Una ontología define los términos básicos y las relaciones que componen el vocabulario de un área temática, así como las reglas para combinar dichos términos y relaciones con el fin de definir extensiones del vocabulario" [22].

"Es una especificación explícita de una conceptualización" [13].

"Un conjunto de axiomas lógicos diseñados para explicar el significado pretendido de un vocabulario" [14].

Por otro lado Cabré [7] define una terminología, dentro de las diferentes disciplinas científico-técnicas, como el conjunto de las unidades de expresión y comunicación que permiten transferir y comunicar el pensamiento especializado. Una terminología persigue el objetivo de fijar unas unidades terminológicas como formas normalizadas y de referencia que descartan las demás variantes para denominar un mismo concepto con el fin de alcanzar una comunicación profesional precisa, moderna y unívoca.

Para finalizar este apartado haremos referencia a Bodenreider [6] afirmando lo siguiente: "Aunque existen más de sesenta sistemas terminológicos en el ámbito biomédico, pocos son clasificables como ontologías". Sin embargo también añade (haciendo mención a *SNOMED-CT*): "Curiosamente la mayoría de los sistemas recientes, tienden a ser ontologías".

Es a éste último sistema y a *UMLS* a los que va dedicado el resto del punto actual.

2.2.1. *SNOMED-CT*

SNOMED-CT o *Systematized Nomenclature of Medicine Clinical Terms*[1] es una extensa terminología clínica de atención médica, disponible en varios idiomas y usada en la actualidad por más de cincuenta países. Nace de la fusión entre *SNOMED RT* desarrollada por el *College of American Pathologists* (*CAP*) y el *Clinical Terms Version 3* (*CTV3*) desarrollada por el *National Heath Service* (*NHS*) de Reino Unido.

En 2007 los derechos de propiedad intelectual fueron transferidos a la *International Health Terminology Standards Development Organisation* (*IHTSDO*) quien se encarga de su mantenimiento y distribución hoy en día.

2.2.1.1. Componentes

Los principales componentes de *SNOMED-CT* son los conceptos, las descripciones y las relaciones.

Conceptos

Los conceptos representan como su propio nombre indica conceptos o ideas clínicas. Cada uno tiene un identificador numérico único (*ConceptID*). Están organizados en jerarquías que van de lo general a lo específico a medida que se desciende en ellas.

Descripciones

Un concepto puede tener más de una descripción asociada, cada una representando un sinónimo que describe la misma idea.

[1] http://www.ihtsdo.org/snomed-ct/

Relaciones

Las relaciones permiten asociar a un concepto otros conceptos cuyo significado está relacionado. La relación más importante es la llamada "es un" que define la mencionada jerarquía entre conceptos. Otros ejemplos son "agente causal", "sitio de hallazgo" o "morfología asociada". Existen cuatro tipos de relaciones: definitorias, calificadoras, históricas y adicionales.

2.2.2. *UMLS*

UMLS o *Unified Medical Language System*[2] es un conjunto de archivos y software que reúne gran cantidad de vocabularios biomédicos y de salud, y los estándares que dan la posibilidad de que distintos sistemas informáticos operen entre sí. Fue desarrollado por la *National Library of Medicine* (*NLM*) tratando de superar dos importantes barreras relativas a la recuperación efectiva de información por parte de máquinas:

- La primera de ellas es la variedad de formas en que unos mismos conceptos se expresan en diferentes fuentes legibles por máquinas y personas.
- La segunda es la falta de un formato estándar para distribuir y difundir terminologías.

2.2.2.1. Estructura

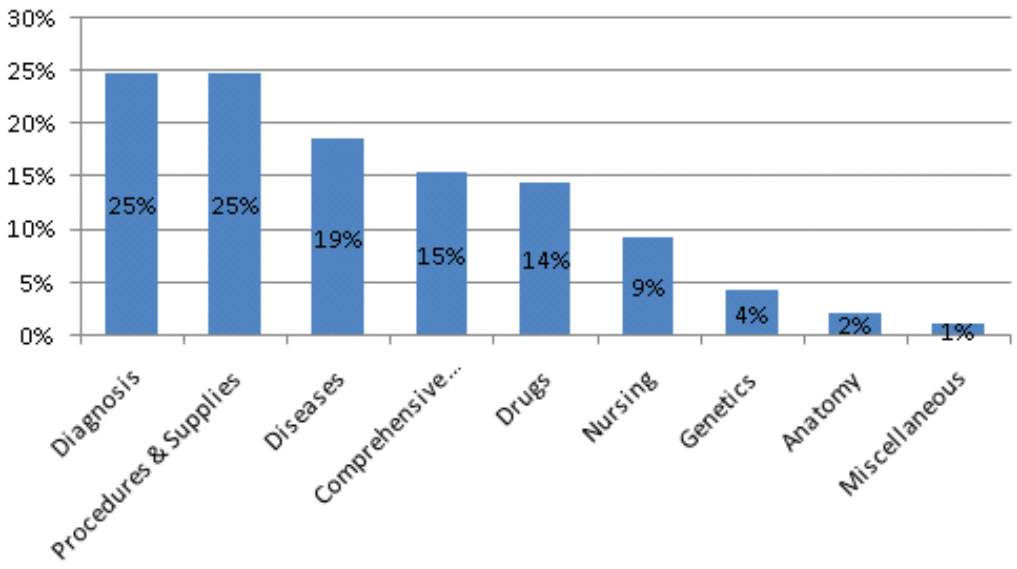
Existen tres principales fuentes de conocimiento de *UMLS:* el *Metatesauro*, la *Red Semántica* y el *Lexicón Especializado y Herramientas Léxicas*.

Metatesauro

El Metatesauro[3] es una base de datos de vocabulario multi-propósito y multi-lenguaje que contiene información sobre conceptos relacionados con la biomedicina y la salud, sus distintos nombres y las relaciones entre ellos. Engloba más de un millón de conceptos biomédicos y más de 100 fuentes de vocabulario, entre las que se encuentran *MeSH*[4], *RxNorm*[5] y *SNOMED-CT*[6]. No es un vocabulario, comprende muchos vocabularios estándar y sirve de apoyo a la hora de crear asignaciones entre ellos, pero no pretende reemplazarlos. Existen varias categorías para clasificar los vocabularios. Algunos pueden pertenecer a más de una categoría. En la figura 2.1 se muestra qué porcentaje representa cada una de las categorías en el Metatesauro.

Cuando un concepto es añadido al Metatesauro recibe un identificador único y es situado en su estructura. Esta estructura tiene cuatro niveles de especificación:

[2] http://www.nlm.nih.gov/research/umls/
[3] http://www.nlm.nih.gov/research/umls/knowledge_sources/metathesaurus/index.html
[4] http://www.nlm.nih.gov/mesh/meshhome.html
[5] http://www.nlm.nih.gov/research/umls/rxnorm/index.html
[6] http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html



Fuente www.nlm.nih.gov

Figura 2.1: Porcentaje que representa cada una de las categorías del Metatesauro *UMLS*.

1. ***Concept Unique Identifiers (CUI)*** o identificadores únicos de concepto. Estarán formados por la letra C seguida de siete dígitos. Un concepto es un significado. Un significado a su vez puede tener diferentes nombres. Los objetivos clave son comprender el significado del nombre en cada fuente de vocabulario y vincular todos los sinónimos. Un *CUI* por tanto se asocia con los denominados conceptos, cada conjunto de conceptos sinónimo estará agrupado bajo el mismo *CUI*.

2. ***Lexical Unique Identifiers*** (***LUI***) o identificadores únicos de léxico. Estarán formados por la letra L seguida de siete dígitos. Enlazan cadenas que son variantes léxicas. Dichas variantes léxicas son detectadas utilizando el *Lexical Variant Generator* (*LVG*) una de las herramientas de *UMLS*. Un *LUI* se asocia con los llamados términos, un conjunto de nombres normalizados tendrá asignado el mismo *LUI*.

3. ***String Unique Identifiers*** (***SUI***) o identificadores únicos de cadena. Estarán formados por la letra S seguida de siete dígitos. Cada nombre de concepto único en cada lenguaje del Metatesauro tendrá asociado un *SUI*. Cualquier variación en el conjunto de dichos caracteres supone una cadena diferente con un *SUI* diferente.

4. ***Atom Unique Identifiers*** (***AUI***) o identificadores únicos de átomos. Estarán formados por la letra A seguida de siete dígitos. Son el componente básico del la estructura y cada vez que se presenta una nueva cadena en un vocabulario, a esta se le asigna un *AUI*. Si la misma cadena aparece varias veces en el mismo vocabulario como un nombre alternativo para diferentes concepto, un *AUI* único será asignado para cada caso. Un *AUI* se asocia con cada nombre de concepto de cada fuente determinada.

A continuación proponemos un ejemplo, acompañado por la figura 2.2, para facilitar la compresión de lo anteriormente explicado.

- En el ejemplo propuesto contamos con cinco cadenas de caracteres aportadas por cuatro fuentes de vocabulario (*BI*, *SNOMED*, *MeSH* y *DX*). Tal como es de esperar, se puede comprobar cómo a cada nombre de concepto de cada fuente de vocabulario determinada, se le asocia un *AUI*.

- Anteriormente hemos mencionado que cada nombre de concepto (teniendo en cuenta cualquier variación en los caracteres) tendrá asignado un *SUI*. En este caso, contamos con cuatro nombres diferentes: *headaches*, *Headache*, *Cranial Pain* y *HEAD PAIN CEPHALGIA*, obteniendo así cuatro identificadores únicos de cadena.

- Así pues, pasamos al nivel inmediatamente superior en la estructura, el *LUI*. Dado que bajo un mismo *LUI* se agrupa un conjunto de nombres normalizados, tendríamos tres *LUI*. El primero será para la normalización de los nombres *head* y *Headache*, el segundo para la normalización de C*ranial Pain* y el último para la normalización de *HEAD PAIN CEPHALGIA*.

- Por último hemos dicho que bajo un *CUI* se agrupan conceptos sinónimo, significados. Así pues para los dos *LUI* mencionados anteriormente seria asignado un único *CUI*, *Headache*.

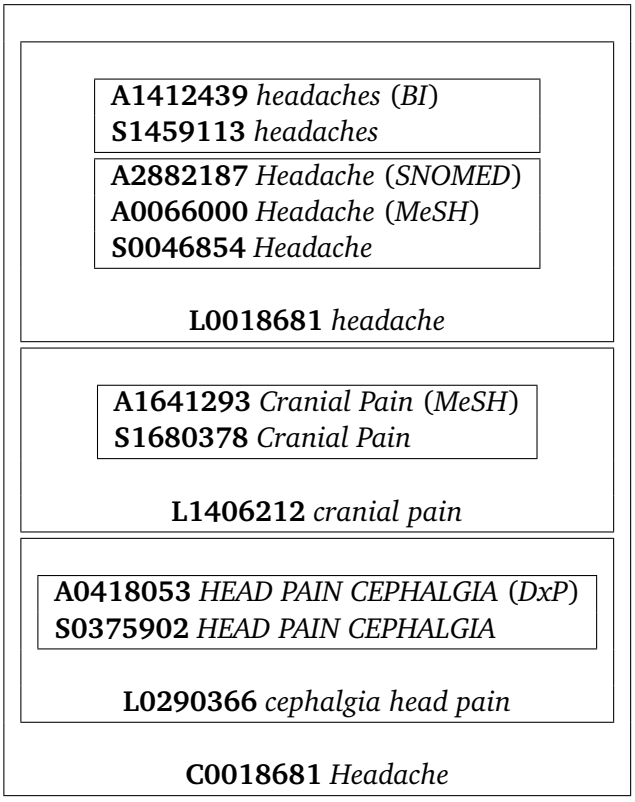


Figura 2.2: Identificadores a lo largo de la estructura del Metatesauro para el concepto *Headache Fuente: www.nlm.nih.gov*

Por otro lado es importante destacar que, debido a su volumen, para poder utilizar de manera efectiva el Metatesauro en una aplicación local, debemos personalizarlo limitando los vocabularios, lenguajes relaciones o atributos a usar, ya que no será habitual que

requiramos los servicios de todos ellos. Para este fin debemos utilizar MetamorphoSys, del que hablaremos más adelante.

Red Semántica

La Red Semántica[7] puede utilizarse para clasificar el vocabulario médico y se compone a su vez por los tipos semánticos y las relaciones semánticas.

Tipos Semánticos Actualmente existen 133 tipos semánticos dispuestos en una jerarquía que tiene dos tipos principales: *Entity* y *Event*. Además están organizados jerárquicamente. La información asociada a un tipo semántico incluye:

- un identificador único;
- tres números que indican su posición en la jerarquía "*is a*";
- una definición;
- sus hijos y padre inmediatos.

Cada concepto del Metatesauro tiene asignado al menos un tipo semántico, que será lo más específico posible.

A continuación, en la figura 2.3 mostramos algunos de los tipos semánticos.

Entity	*Event*

▪ *Physical Object*
  • *Organism*
  ○ *Plant*
  ○ *Fungus*
  ○ *Virus*
  ○ ...
  • ...
▪ *Conceptual Entity*
▪ ...
	▪ *Activity*
	  • *Behavior*
	  ○ *Social Behavior*
	  ○ *Individual Behavior*
	  • *Daily or Recreational Activity*
	  • *Occupational Activity*
	  • *Machine Activity*
	▪ *Phenomenon or Process*
	▪ ...

Figura 2.3: Fragmento de la jerarquía de Tipos Semánticos

[7] http://semanticnetwork.nlm.nih.gov/

Relaciones Semánticas Actualmente existen 54 relaciones semánticas entre los tipos de la red semántica, y la que enlaza la mayoría de los conceptos es la relación "*is a*". Esta relación permite establecer la jerarquía de tipos en la Red Semántica y obtener el concepto más específico a asignar a cada concepto del Metatesauro. De esta manera podríamos afirmar por ejemplo que: *Animal is a Entity* o que *Virus is a Organism*.

Un punto a destacar es que las relaciones semánticas se establecen entre tipos semánticos, pero puede que no todos los conceptos que tengan asignados dichos tipos semánticos mantengan la relación.

En la figura 2.4 mostramos de nuevo un fragmento con algunas de las relaciones existentes.

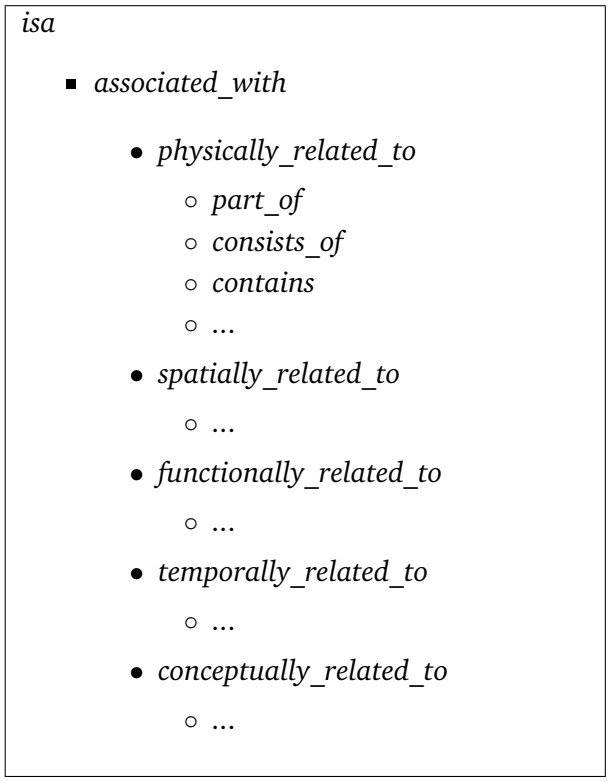


Figura 2.4: Fragmento del conjunto Relaciones Semánticas

Lexicón Especializado y Herramientas Léxicas

El Lexicón Especializado[8] disponible únicamente en inglés es un diccionario que incluye más de 200.000 términos biomédicos y palabras inglesas de uso frecuente. Estos términos son recuperados de diversas fuentes y utilizados por las herramientas léxicas para servir como ayuda en el PLN.

La entrada de cada palabra o término en el lexicón incluye información que puede ser de alguno de los siguientes tipos:

- sintáctico;

[8] http://lexsrv3.nlm.nih.gov/Specialist/Summary/vtt.html

- morfológico;
- ortográfico.

Las Herramientas Léxicas son un conjunto de programas que, como ya hemos mencionado, tienen como fin servir de apoyo en el PLN. Las principales son:

- el generador de variantes léxicas (*LVG*);
- el generador de cadenas de caracteres normalizadas (*Norm*);
- el generador de índices de palabras (*WordInd*).

Estas herramientas se encargan conjuntamente de abordar el problema de la variabilidad de las palabras y términos, por ejemplo aquellas formas variadas que pueden surgir al conjugar verbos, de manera que el usuario pueda abstraerse de esta situación.

2.2.2.2. MetamorphoSys

MetamorphoSys[9] es una herramienta multi-plataforma que la *NLM* actualiza e incluye en cada publicación de *UMLS*. Su finalidad es dar al usuario la posibilidad de creación de un subconjunto personalizado del Metatesauro, siendo además necesaria para instalación de manera local de las fuentes de conocimiento de *UMLS*.

Podemos estar interesados en trabajar con subconjuntos de Metatesauro debido a dos motivos principales:

- El primero se deriva del tamaño del propio Metatesauro: no será habitual que requiramos el uso de toda la información almacenada originalmente en él. Por otro lado, ciertas fuentes de vocabulario requieren licencias especiales para usos específicos y puede que no estemos interesados en adquirir dichas licencias.
- El segundo es que el usuario puede querer modificar el formato de los datos de salida o aplicarles diversos filtros.

Esta personalización requerirá sin embargo un gran conocimiento de los requisitos de nuestra aplicación y del Metatesauro. Los filtros existentes en la actualidad son:

- *Attributes Type List*
- *Content View List*
- *Languages List*
- *Relations List*
- *Sample Meta*
- *Semantic Types List*

[9] http://www.nlm.nih.gov/research/umls/implementation_resources/metamorphosys/index.html

- *Source Subset List*
- *Source Term Types List*
- *UI List*
- *Unicode*

La salida proporcionada por MetamorphoSys consistirá en un conjunto de archivos *Rich Release Format* (*RRF*) u *Original Release Format* (*ORF*). Además al crear un conjunto del Metatesauro o instalar la Red Semántica se podrán generar *scripts* para ser cargados por *MySQL*, *Oracle* o *Microsoft Access*.

2.3. MetaMap

La identificación de conceptos supone un punto fundamental en la realización de numerosas tareas como la recuperación de información, la clasificación, o la obtención de resúmenes.

MetaMap[10] es un software desarrollado por el Dr. Alan Aronson en la *NLM* de Estados Unidos que cuenta con múltiples opciones de configuración y permite mapear o asociar términos que aparecen en un texto biomédico, con conceptos del Metatesauro *UMLS*. Utiliza enfoque basado en el PLN y en técnicas lingüísticas.

Además de ser aplicado tanto para recuperación de información (*IR*) y aplicaciones de minería de datos es uno de los fundamentos del *Medical Text Indexer* (*MTI*) del *NLM* el cual se utiliza para indexar de manera semiautomática o automática literatura del *NLM*.

2.3.1. Funcionalidad

Entre la funcionalidad ofertada por MetaMap nos interesa especialmente la relativa a los puntos que se presentan a continuación, y que es descrita por Aronson and Lang [3].

2.3.1.1. Desambiguación

Uno de los problemas principales que afectan al PLN es la ambigüedad del lenguaje, y una de las mayores debilidades de MetaMap es su incapacidad para resolver la ambigüedad del Metatesauro, es decir las situaciones en la que dos o más conceptos comparten un sinónimo. Es por ello que se incluyó finalmente un sistema de desambiguación o *Word Sense Disambiguation* (*WSD*) que se puede activar mediante el uso de la opción `-y`. Cabe mencionar que además se favorecen aquellas alternativas que tienen un tipo semántico más probable en ese contexto determinado.

[10] http://metamap.nlm.nih.gov/

2.3.1.2. Detección de negaciones

MetaMap es capaz de hacer uso de una versión extendida del algoritmo NegEx[11] para poder determinar cuándo un concepto está negado. Para ello, si se está utilizando el formato de salida por defecto (*human-readable*) es necesario hacer uso de la opción `-negex`. Para conocer más detalles sobre esta característica recomendamos al lector dirigirse al apartado Section 2.7.

2.3.1.3. Detección de acrónimos y abreviaturas definidos por el autor

En multitud de textos de dominio técnico aparecen con frecuencia acrónimos y abreviaturas (*AA* del inglés *acronyms and abbreviations*) que además suelen ir junto a sus definiciones o extensiones. De esta manera interesa que cuando un acrónimo o sigla haya sido definido, esa misma definición sea la que se asigne en futuras apariciones. Para detectar *AA* MetaMap aplica un algoritmo que es fundamentalmente el mismo que se describe en Schwartz and Hearst [29]. Se trata de asociar lo que potencialmente es un *AA*, que deberá estar escrito entre paréntesis, con su supuesta expansión, que debe estar situado precediendo al propio *AA* dentro de la misma frase.

Existen ciertas reglas que es importante mencionar para no encontrarse con un funcionamiento inesperado, y evitar intentos infructuosos:

- Los *AA* no pueden contener más de 20 caracteres.
- Las expansión deben ser mayores que los *AA* correspondientes.
- Una expansión no puede contener texto entre paréntesis.
- Cada palabra considerada como *AA* debe contener como mucho 12 caracteres.
- Los *AA* no pueden comenzar por *such*, *also* o *including*.

2.3.1.4. Conceptos

Pero por supuesto la parte esencial de MetaMap es aquella que lo define el mapeo de términos de un texto biomédico a conceptos del Metatesauro *UMLS*.

El algoritmo que sigue para llevar a cabo este proceso engloba diferentes fases que son detalladas por Aronson [1].

1. ***Parsing***

   Haciendo uso del Léxico Especializado realiza un primer análisis sintáctico siendo capaz de detectar diferentes elementos textuales, como la palabra principal de una frase.

[11] http://code.google.com/p/negex/

2. ***Variant Generator***

   Para cada frase se genera una variante utilizando de nuevo el Léxico Especializado y en esta ocasión, de manera complementaria, una base de datos de sinónimos. Dicha variante consistirá en un sintagma nominal junto con variantes ortográficas, sinónimos, acrónimos o abreviaturas entre otros.

3. ***Candidate Retrieval***

   Formar el conjunto candidato de todas las cadenas del Metatesauro que contienen al menos una de las variantes.

4. ***Candidate Evaluation***

   Asignar a cada candidato un sintagma nominal, evaluarlo o puntuarlo y ordenar los candidatos por puntuación.

5. ***Mapping Construction***

   Combinar los candidatos que están involucrados en partes disjuntas del sintagma nominal. Calcular de nuevo la puntuación y seleccionar en base a dicha puntuación [2].

2.3.2. Salida generada

MetaMap puede generar archivos con diferentes formatos de salida:

- ***Human-Readable***: es el formato de salida por defecto y muestra, para cada frase del texto de entrada, la propia frase, una lista de conceptos candidatos del Metatesauro asociados a cada parte de la frase, el mapeo formado al realizar las combinaciones de candidatos asociados a partes disjuntas, y datos complementarios como la puntuación otorgada, u otros opcionales como el *CUI* del concepto (`-I`), los tipos semánticos (`-s`), o las fuentes (`-G`).

- ***MetaMap Matching Output* (*MMO*)**: incluye un *súper-conjunto* de la información de la salida *Human-Readable* y tiene formato de términos *Prolog*. Su utilidad radica en que así, se permitiría realizar un pos-procesamiento por parte de aplicaciones *Prolog*. Para obtener este tipo de salida, debemos hacer uso de la opción `-q`.

- **XML**: debido a la importancia del `.xml`, también existe la posibilidad de obtener, en este formato, los mismos datos que se presentan en la salida *MMO*. Esto se puede conseguir mediante las opciones `--XMLf --XMLf1 --XMLn --XMLn1`. La `f` indica que el `.xml` tendrá cierto formato mientras que la `n` indica que no lo tendrá. El `1` señala que se generará un `.xml` para *un* archivo de entrada, mientras que el hecho de no incluirlo, nos quiere decir que obtendremos uno por cada entrada citada. A pesar de las ventajas que puede presentar, tiene el inconveniente de que los archivos resultan considerablemente pesados en disco.

- ***Colorized MetaMap Output* (*MetaMap 3*D)**: este formato está diseñado para proporcionar de manera visual y mediante colores información para los conceptos mapeados por MetaMap en un texto.

- ***Fielded MetaMap Indexing (MMI) Output***: la opción `-f` proporciona una salida con múltiples líneas que contienen campos delimitados por tabulaciones. Su contenido es el mismo que el de la salida *MMO*. Este tipo de salida es utilizado principalmente por el *Medical Text Indexer* (*MTI*).

2.3.3. MetaMap API

MetaMap API[12] es una librería en *Java* desarrollada por Willie Rogers que permite la comunicación de un proyecto desarrollado en Java con el servidor de MetaMap para servirse de toda su potencia. Para su instalación y uso es imprescindible contar con una versión instalada de MetaMap y con *Java 1.6 SDK* o una versión superior. Además los servidores de MetaMap deberán estar en ejecución para la correcta utilización de la librería. Cabe señalar que el motor está escrito principalmente en *SICStus Prolog* y que para facilitar su uso por programas *Java* se utiliza *Prolog Beans*.

2.4. Detección y desambiguación de conceptos médicos

A la hora de detectar un concepto, no es suficiente con darse cuenta de que el conjunto de palabras que lo compone, está almacenado como concepto médico en una base de datos, ya que es posible que un mismo conjunto de palabras pueda tener diferentes interpretaciones en función del contexto en que se encuentre. Siguiendo con el punto anterior sobre ontologías y terminologías biomédicas, pasamos a analizar algunas de las formas en que podemos extraer y desambiguar conceptos médicos de un texto. En primer lugar haremos referencia al sistema en que basamos nuestro trabajo y a continuación pasaremos a comentar algún detalle adicional sobre MetaMap relacionado con este tema.

2.4.1. Procesador automático de informes médicos

El proyecto "Procesador automático de informes médicos" [5], basaba la detección de un concepto médico en la aparición del mismo en una base de datos construida a partir de *SNOMED-CT*. Por otro lado, la desambiguación se realizaba apoyándose en su ubicación en el texto y en las palabras con las cuáles este formaba una frase.

Toda esta información se encontraba en unas bases de datos que fueron completadas usando aprendizaje automático fundamentado en un conjunto, que tenían disponible, de informes médicos en español.

2.4.2. MetaMap

Tal como hemos mencionado anteriormente, MetaMap incorpora desambiguación de conceptos, de manera que los resultados obtenidos suelen ser bastante acertados y próximos a la realidad.

El algoritmo original utilizado para ello fue escrito por Smith, Rindflesch, and Wilbur [31] del *National Center for Biotechnology Information* (NCBI) y del *Lister Hill National*

[12] http://metamap.nlm.nih.gov/README_javaapi.html

Center for Biomedical Communications (*LHNCBC*). Inicialmente se escribió para *C++* y para *Perl* y se modificó de manera que también estuviese disponible para *Java*.

Se basa en el Modelo Oculto de Markov (*Hidden Markov Model* o *HMM*) [34] y se modificó para incluir información contextual junto con información gramatical para mejorar el marcado de los conceptos.

Utilizando información que se obtuvo durante un proceso de entrenamiento, se determina cuál es la probabilidad de aparición de un concepto teniendo en cuenta las palabras por las que está rodeado y de su categoría gramatical [31].

Acorde con la información presente en la precisión del algoritmo *MedPost/SKR* es tal que para un conjunto de 1000 frases, se obtuvo una precisión de $97{,}43\,\%$ con 26566 *tokens*, con 582 correctamente etiquetadas y 261 etiquetadas con un solo error.

2.5. Corrección ortográfica

El procesado de los informes médicos incluye una fase de corrección ortográfica automática y es por ello que fue necesario considerar diversos correctores y distintos algoritmos que permitiesen puntuar las sugerencias en caso de que se detectase una falta ortográfica. En las siguientes sub-secciones se hace mención a las alternativas más destacadas.

2.5.1. Correctores ortográficos

A modo de resumen, podemos decir que un corrector ortográfico se encarga en primer lugar de analizar un texto y extraer todas las palabras que este contiene. Tras ello compara cada palabra con una lista de palabras conocidas y que considera correctas, es decir un diccionario. Este diccionario podría contener únicamente dicha lista de palabras o información adicional, como atributos gramaticales.

Una vez que se ha comprobado que una determinada palabra no aparece en el diccionario utilizado, se obtienen de él una serie de términos similares al incorrecto y se presentan como posibles alternativas o soluciones.

Existen numerosos correctores ortográficos, cuya diferencia radica, general, en el método empleado para obtener las palabras a sugerir. A continuación hablaremos de dos de los correctores ortográficos más importantes cuyo uso se planteó al comienzo del desarrollo, y de las técnicas que utilizan.

2.5.1.1. Aspell

Aspell[13], desarrollado y mantenido por Atkinson [4], es un corrector ortográfico de código abierto con licencia *GNU Library or Lesser General Public License (LGPL)*[14] que nació a partir de Ispell[15] con el objetivo de mejorarlo y reemplazarlo. A diferencia de este, cuyo

[13] http://aspell.net/
[14] http://www.gnu.org/licenses/lgpl.html
[15] http://www.lasr.cs.ucla.edu/geoff/ispell.html

uso se descartó rápidamente, facilita comprobar la ortografía de documentos con codificación UTF-8 sin necesidad de utilizar diccionarios especiales. Una de sus principales contribuciones fue el uso del algoritmo Metaphone Philips [25]. Por otro lado también permite el uso simultáneo de varios diccionarios y se ha comprobado que es capaz de realizar mejores sugerencias. Así por ejemplo para la palabra *trubble*, Ispell sugerirá únicamente *rubble*, donde Aspell sugiere *trouble* (como primera opción) junto con *dribble*, *rubble*, y otras palabras. La desventaja que podría presentar, es que uno de los algoritmos que utiliza es especifico para el inglés, pero dada la naturaleza del proyecto en el que trabajamos esto no supone un problema.

Según Jacquemont et al. [18] el rendimiento de ambos puede compararse utilizando dos criterios principales. El primero, *Suggestion Intelligence* (*SI*), corresponde a la tasa de sugerencias adecuadas propuestas respecto a todas las faltas existentes.

$$SI = \frac{Total\ correct\ suggestions}{Total\ misspellings} * 100$$

El segundo criterio, *Suggestion Intelligence First* (*SIF*) representa la habilidad del corrector para realizar en primer lugar, dentro de la lista de todas las propuestas, una buena sugerencia.

$$SIF = \frac{Total\ correct\ suggestions\ found\ first\ on\ list}{Total\ misspellings} * 100$$

Tras realizar pruebas con Aspell e Ispell para un conjunto de documentos que contenían 500 faltas ortográficas identificadas, obtuvieron los resultados mostrados en el cuadro 2.1.

	Aspell	**Ispell**
SI	79	78
SIF	58	39

Tabla 2.1: Estudio comparativo del estado del arte de correctores ortográficos (2007) *Fuente: Jacquemont et al. [18]*

2.5.1.2. Hunspell

Hunspell[16], además de ser un analizador morfológico, es corrector ortográfico basado en MySpell, escrito en C++ y desarrollado por László Németh que se encuentra bajo licencias *LGPL* y *Mozilla Public License 1.1 (MPL 1.1)*[17]. Permite utilizar codificación de caracteres Unicode y fue originalmente diseñado para lenguajes con una composición y morfología complejas (como húngaro y alemán). Por otro lado, da la posibilidad de mejorar el sistema de sugerencias mediante el uso de n-gramas y diccionarios que cuentan con datos basados en la pronunciación, y además cuenta con características adicionales que permiten indicar afijos, sufijos y palabras prohibidas.

Hunspell es utilizado por numerosas y conocidas aplicaciones informáticas como Eclipse, LyX, Libre Office, *Evernote*, *Mozilla Thunderbird*, *Mozilla Firefox*, *Google Chrome* o el sistema operativo de *Apple Mac OS X* (a partir de la versión 10.6), sustituyendo a Aspell

[16] http://hunspell.sourceforge.net/
[17] http://www.mozilla.org/MPL/1.1/

en algunos casos, y asimismo existen *ports* para *Delphi*, *Java*, *Perl*, *Python* y *Ruby* entre otros.

A pesar de que podemos encontrar estudios[18] que indican que los resultados de Aspell eran mejores que los de Hunspell para el inglés, podemos ver desde la propia página, que a pesar de que en los últimos años se ha intentado evitar esta situación, con el tiempo Aspell se ha ido estancando.

2.5.2. Comparación de cadenas de caracteres

Las métricas de comparación de cadenas permiten calcular la diferencia existente entre dos cadenas de caracteres. Dado que nos interesa realizar una fase de corrección ortográfica, nos interesará por tanto poder comparar una cadena de caracteres que representa una palabra escrita de manera incorrecta con la sugerencia propuesta por el corrector ortográfico. Durante el resto de la sección hablaremos de algunas de las métricas más conocidas. Cabe mencionar que el proyecto del que partimos [5], también tuvo en cuenta estas métricas, pero, puesto que nuestro proyecto pretende ser válido tanto para inglés como para español, las elecciones variarán ligeramente.

Volviendo al punto anterior, es importante notar que la diferencia entre las dos cadenas a comparar puede basarse en distintas características (por ejemplo, un algoritmo sencillo podría consistir en calcularla tomando únicamente la diferencia de longitud existente entre ambas). Lo más común es obtenerla considerando que el objetivo es transformar una en otra mediante diversas operaciones, que tendrán un coste o penalización asociado. Según se apliquen unas u otras, y teniendo en cuenta el coste de cada una de ellas, la distancia aumentará en mayor o menor medida, sabiendo que siempre buscamos el coste mínimo.

En función de cuál sea este conjunto de transformaciones posibles y de cuál sea su penalización, podemos encontrar diferentes algoritmos.

2.5.2.1. Distancia de Hamming

En el ámbito de la teoría de la información se conoce como distancia de Hamming Hamming [15] entre dos cadenas de igual longitud al número mínimo de sustituciones de caracteres que debemos realizar para convertir una en otra. O visto de otra forma, es equivalente a la cantidad de caracteres distintos entre ambas. De esta manera si tenemos las cadenas de texto

- **c**a**b**eza
- **m**a**l**eza

entonces la distancia de Hamming entre ambas es 2.

[18] http://aspell.net/test/cur/ *Consultada actualización de 2008*

```java
public int hammingDistance(String cadena1, String cadena2) {
   if (cadena1.length() != cadena2.length())
      return -1;
   int distancia = 0;
   for (int i = 0; i < cadena1.length(); i++)
      if (cadena1.charAt(i) != cadena2.charAt(i))
         distancia++;
   return distancia;
}
```

Listado de código 2.1: Distancia de Hamming en *Java*

2.5.2.2. Distancia de Levenshtein

La distancia de Levenshtein [20], también conocida como distancia de edición, es una de las más populares. Recoge como operaciones básicas de transformación las siguientes, todas ellas con coste asociado 1:

1. Inserción de un carácter
2. Borrado de un carácter
3. Sustitución de un carácter

Dado que se tienen en cuenta las operaciones de inserción y eliminación, no es necesario que ambas cadenas de texto tengan la misma longitud, como sucedía en el caso anterior. De nuevo, como es de suponer, buscaríamos la distancia mínima, es decir el número de transformaciones mínimo. De esta manera para las cadenas "hola" y "ola" bastaría con insertar una 'h' al comienzo, de manera que la distancia resultante sería 1. Una de las implementaciones de este método viene dado por el algoritmo descrito por Wagner and Fischer [33], y fue el utilizado por el proyecto "Procesador automático de informes médicos" desarrollado por [5].

2.5.2.3. Distancia de Damerau-Levenshtein

Para calcular la distancia de Damerau-Levenshtein [10] existente entre dos palabras, se consideran según esta técnica, las mismas operaciones que en el caso anterior, y además una adicional que permite el intercambio de dos caracteres adyacentes, también con coste 1.

Según Damerau [10], considerando que una falta ortográfica como aquella situación en la que las palabra incorrecta se encuentra a distancia 1 de la palabra correcta, suponiendo estas reglas, estamos cubriendo el 80 % de las faltas ortográficas más comunes.

Complicando algo el caso anterior, podríamos encontrar "hola" y "loas" de manera que mediante los siguientes cambios:

1. **lo**as - o**l**as (Intercambio de caracteres adyacentes).
2. olas **- h**olas (Inserción de un carácter).

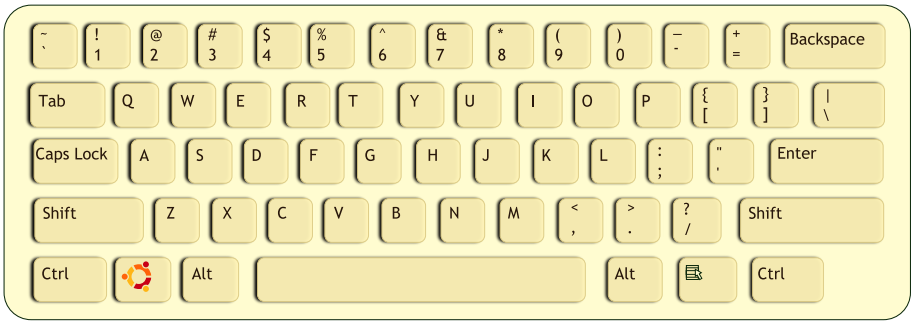
3. hola**s** - hola (Supresión de un carácter).

Obtenemos como distancia entre ambas cadenas 3.

Cabe mencionar que esta técnica, no solo se ha utilizado para la implementación de correctores ortográficos, sino que también ha sido empleada para en biología para realizar mediciones con ADN.

2.5.2.4. Distancia de teclado y algoritmo de Needleman-Wunsch

Otra distancia a tener en cuenta, es la distancia de teclado, que supone que es más probable que los errores ortográficos estén producidos de tal manera que una buena sugerencia pasa por considerar los caracteres cercanos en un teclado QWERTY (figura 2.5). La elección de este tipo de teclado (denominado así por la colocación de las letras de la primera fila) se debe a que en la actualidad es uno de los más usados. Utilizar esta distancia sería útil por ejemplo suponiendo la distancia de Levenshtein y contando por tanto con la operación de sustitución de caracteres. Así, ante la detección de la palabra "me**a**a" falta ortográfica podríamos encontrar como sugerencias las palabras "me**s**a y "me**t**a", sin embargo atendiendo a esta técnica, sería más acertado proponer como palabra correcta "mesa" y que el las teclas que representan la S y la A están contiguas en el teclado QWERTY, mientras que la T está mas alejada.



Fuente: http://openclipart.org/detail/31855/qwerty-keyboard-%28path%29-by-rockraikar

Figura 2.5: Teclado QWERTY

Es por esto, que interesaría medir la distancia entre caracteres y finalmente entre cadenas de texto siguiendo este método. Con el objetivo de calcular la distancia entre caracteres una posibilidad bastante sencilla sería utilizar la distancia de Manhattan considerada por Hermann Minkowski en el siglo XIX [19] y también conocida como métrica *Taxicab*.

Sin embargo para poder aplicarla correctamente, el proyecto en que nos basamos hizo uso del algoritmo de Needleman and Wunsch [23] en los siguientes párrafos explicaremos en qué consiste este algoritmo y el porqué de su uso.

El algoritmo de programación dinámica de Needleman-Wunsch, originalmente utilizado en el ámbito de la bioinformática para comparar secuencias de proteínas, ADN o ARN, sirve para alinear globalmente dos secuencias [36] de manera que se maximice el número de elementos que, estando en la misma posición en ambas cadenas, son coincidentes. Para conseguir este alineamiento se cuenta con una operación de inserción de huecos

entre posiciones. Tal como apuntaron Barahona et al. [5], ha sido demostrado que este problema es equivalente al de la minimización de distancia de edición por Sellers [30].

De esta manera se alinean las secuencias conociendo los alfabetos que permiten formarlas y la diferencia existente entre cada uno de los símbolos que los componen. Para especificar esta diferencia que actuará como penalización, y suponiendo que los alfabetos A y B tienen tamaños m y n respectivamente, se utilizará una matriz de $m * n$ de manera que la posición $[i,\ j]$ de la matriz especifique la penalización a aplicar en caso de que estén alineados los símbolos $i - ésimo$ de A y $j - ésimo$ de B. Además será preciso indicar cuál es la penalización que se debe aplicar en caso que un símbolo quede alineado con un hueco.

Dicho esto, podemos ver por qué nos interesa. Esto es, porque dadas dos cadenas de caracteres y la distancia existente entre cada uno de ellos, sirviéndonos del teclado QWERTY y de la distancia de Manhattan para formar la matriz, podemos obtener el alineamiento óptimo que nos permitirá saber cómo de similares son dichas cadenas.

2.5.2.5. Algoritmos fonéticos

Otro de los métodos de comparación de cadenas que puede resultar de interés en la corrección ortográfica, es aquel que se centra en obtener una aproximación de representación fonética de estas con el fin de comparar unas con otras. Esta representación será, evidentemente dependiente del idioma, pero existen multitud de algoritmos dedicados a ello, y especialmente para el inglés. Algunos de los más conocidos son los que presentamos brevemente a continuación.

Soundex

Soundex Russell and Odell [28] es un algoritmo fonético implementado con el fin de obtener una representación fonética aproximada de apellidos en inglés. De esta manera podría decirse de dos de ellos, que se escribiesen de forma distinta pero que tuviesen una pronunciación *similar*, que son equivalentes, puesto que tendrían asociada una misma clave o código. Dicho código estará compuesto por una letra y tres números. La letra, será la primera del apellido y los tres números serán dados en función de cuáles sean las tres letras siguientes, atendiendo a la codificación de el cuadro 2.2.

En caso de que el apellido tenga menos de cuatro letras, se añadirán ceros, y en caso de que contenga más de cuatro, el resto de letras serán ignoradas. Cabe señalar, que tal como vemos en el cuadro, las vocales y las letras 'H', 'W' e 'Y' no se tienen en cuenta.

De esta manera *"Washington"* vendría representado como $W252$ y *"Lee"* como $L000$ [17]. Este algoritmo es uno de los más populares, entre otros motivos, por estar incluido como parte de *Oracle* [24] o *MySQL* MySQL [21].

New York State Identification and Intelligence System (NYSIIIS)

New York State Identification and Intelligence System más conocido como simplemente *NYSIIIS* [32] es un algoritmo que pretendía, realizando un mayor análisis de las palabras, mejorar *Soundex*. Para ello seguía el algoritmo que describimos a continuación.

Número	Letras que representa
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Tabla 2.2: Códigos asociados a las letras según *Soundex*

1. Se traducen los primeros caracteres: $MAC \rightarrow MCC$, $PH \rightarrow FF$, $PF \rightarrow FF$, $KN \rightarrow NN$, $K \rightarrow C$, $SCH \rightarrow SSS$.
2. Se traducen los últimos caracteres: $EE \rightarrow Y$, $IE \rightarrow Y$, DT, RT, RD, NT, $ND \rightarrow D$
3. Se toma como primer carácter del código la primera letra.
4. Se traducen letra a letras las restantes, a través de las siguientes reglas:
   a) $EV \rightarrow AF$ si no A, E, I, O, $U \rightarrow A$
   b) $Q \rightarrow G$, $Z \rightarrow S$, $M \rightarrow N$
   c) $KN \rightarrow N$ si no $KN \rightarrow C$
   d) $SCH \rightarrow SSS$, $PH \rightarrow FF$
   e) $H \rightarrow$ en caso de que la anterior o la siguiente no sea una vocal, la anterior.
   f) $W \rightarrow$ en caso de que la anterior sea una vocal, la anterior.
5. Para los últimos caracteres:
   a) Si es S, elimminarlo.
   b) Si es el conjunto AY, sustituirlo por Y.
   c) Si es A, eliminarlo.

Finalmente se concatena el código traducido con el valor obtenido en el paso 3. En caso de que la longitud total sea mayor que 6, se trunca el código dejando únicamente las 6 primeras letras (algunas versiones permiten que la longitud sea mayor, omitiendo este último paso).

A pesar de que *NYSIIS* consigue aumentar en un $2{,}7\,\%$ la exactitud de *Soundex*, también cuenta con algunos problemas relacionados con la posición de las letras [17].

Metaphone

Metaphone [25] es un algoritmo que surgió como propuesta para resolver y paliar algunas de las deficiencias que presentaba *Soundex*. Nació con el fin de representar las palabras según su pronunciación en inglés, como *Soundex*, de manera que aquellas con pronunciación *similar* tuviesen una misma representación, pero en este caso la longitud del código

que representa cada palabra es variable. Podemos encontrar este algoritmo ya incluido como función en algunos sistemas como *PHP* [27]. Sin embargo posteriormente el mismo autor publicó una nueva versión conocida como *Double Metaphone [26]* que mejoraba la anterior. El calificativo *Double* pretendía hacer referencia al hecho de que las palabras o apellidos, podían tener dos códigos, uno primario y otro secundario en lugar de solamente uno.

A pesar de que ambas versiones son gratuitas, en 2009 Philips publicaría una tercera, *Metaphone 3*, que sería comercial. Con esta versión la exactitud mejora desde el 89 % que se lograba alcanzar con *Double Metaphone* hasta más del 98 %. Además cabe señalar que se encuentra disponible para *C++*, *C#*, *Java*, *PHP*, *PL/SQL*[19] y para español y alemán además de inglés.

2.6. Expansión y desambiguación de acrónimos

Los acrónimos y abreviaturas son muy comunes en la terminología médica por lo que es habitual que estos estén presentes en los informes médicos con los que vamos a tratar. Con el fin de que los conceptos puedan ser reconocidos posteriormente será necesario expandir los acrónimos. Además debido al gran número de acrónimos médicos que hay tanto en la lengua española como en la inglesa un mismo acrónimo puede tener más de una expansión por lo que es necesario realizar una desambiguación para saber cuál es la correcta.

2.6.1. Procesador automático de informes médicos

Este sistema contaba con expansión y desambiguación de acrónimos en español, y para tanto uno como otro proceso a cabo se seguían los pasos que indicamos a continuación:

1. División el texto en palabras y búsqueda de cada una de ellas en un lexicón de acrónimos. En caso de que la palabra se encuentre en él será clasificada como acrónimo.
2. En caso de que el acrónimo tenga asociada más de una expansión posible, será necesario decidir cuál se debe utilizar para lo que se utiliza el siguiente método:
   a) El sistema cuenta con una serie de reglas cada una con una expansión asociada, de manera que, en función de qué condiciones se cumplan se escogerá una u otra expansión. Dichas condiciones se refieren a la sección o campo del informe en que se encuentra el acrónimo, o a las palabras del contexto en que aparece el mismo.
   b) En caso de que no se cumpla ningún conjunto de condiciones que permita aplicar las reglas y escoger una expansión, se pasa al siguiente método de desambiguación. El sistema contiene un módulo de aprendizaje que le da la posibilidad de aprender información acerca de en qué contexto suele aparecer cada acrónimo. De esta forma que al tener que elegir entre las distintas desambiguaciones para un acrónimo se compara su situación con los resultados obtenidos durante el aprendizaje y escogiendo así la expansión más adecuada para el contexto en el que aparece el acrónimo.

[19] http://www.amorphics.com/index.html

2.6.2. MetaMap

Ya se ha mencionado en el apartado 2.3.1.3 que MetaMap permite expandir acrónimos y abreviaturas. A continuación damos más detalles acerca de la manera en que esto puede ser posible y de las alternativas que ofrece.

Archivo UDA

MetaMap ofrece la posibilidad de añadir un archivo llamado UDA en el que se añadan acrónimos y sus expansiones, si al procesar un texto MetaMap se encuentra con alguno de los acrónimos existentes en el UDA lo cambiará automáticamente por la expansión que le corresponda.

Acrónimos auto-anotados

Además MetaMap también detecta los acrónimos auto-anotados. Esto quiere decir que si el redactor del texto escribe junto a la expansión de un acrónimo, y entre paréntesis, el acrónimo asociado, entonces en caso de que vuelva a encontrarse dicho acrónimo, este será sustituido por su expansión.

2.7. Detección de negaciones

Durante el proceso de extracción de información de los informes médicos, es muy importante detectar cuándo una frase está afirmada o negada, dado que no es lo mismo afirmar la presencia de un mal o enfermedad que negar la presencia de la misma. Es por ello que a continuación mencionamos algunas de las técnicas que podríamos utilizar para tal fin.

2.7.1. NegEx

El algoritmo NegEx está desarrollado por Chapman, Bridewell, Hanbury, Cooper, and Buchanan [8] y se basa en la utilización de expresiones regulares para la detección de cuando una frase está negada, que palabra de la frase provoca la detección y el ámbito de la frase.

Este algoritmo está especialmente orientado para el campo de la medicina y en inglés y acorde con lo indicado en este artículo científico , tiene una precisión media en torno al $85\,\%$, por lo que sus resultados son muy buenos para tener en cuenta.

2.7.2. MetaMap

Ya se indicó en la sección 2.3 que MetaMap incluye en su núcleo una versión del algoritmo de NegEx modificado. Inicialmente, el algoritmo se encontraba sin ningún tipo de cambio, pero en el año 2009 se decidió ampliar y mejorar con el fin de aumentar la precisión del mismo.

Las modificaciones realizadas fueron, entre otras[20]:

- La inclusión de nuevas palabras clave usadas durante la detección de la negación como: *other than*, *otherwise, to account for, to explain* y *then*.
- Se incorporaron a las bases de datos de las que se nutre MetaMap ciertos conceptos cuyo significado ya representa una negación de un concepto médico, de tal manera que si se encuentra en una frase alguno de ellos, se supondrá que la frase está negada.
- Fusión de diferentes conceptos negados en uno único con el fin de evitar el reconocimiento de un concepto negado varias veces por frase.
- Incorporación de dos tipos semánticos específicos para la negación: *patf* y *neop* .

2.8. Detección de especulaciones

Una de las directrices de futuro indicas en el proyecto previo a este [5], era la inclusión de la detección de frases especuladas, es decir frases en las que el escritor no afirma ni niega los hechos que está narrando.

La detección de este tipo de frases es también muy importante, puesto que en el campo de la medicina, nunca hay certeza absoluta y en muchas ocasiones se utilizan suposiciones, y estas no deben tenerse en cuenta con el mismo peso que una afirmación o una negación.

2.8.1. CoNLL-2010

El CoNLL-2010 es un congreso internacional sobre procesamiento y aprendizaje de lenguaje natural que se desarrolló durante el año 2010.

Una de las tareas propuestas para este congreso fue la detección de especulaciones en textos, así como el ámbito de la frase especulada y las palabras causantes de la detección.

En uno de los artículos publicados en este contexto [12], se explica cuál es el ámbito de una frase especulada en función de ciertas palabras clave en inglés y de su categoría gramatical y finalmente aporta ejemplos de dichas palabras clave.

2.9. Indexación y realización de consultas

Como extensión natural del proyecto surgen la implementación de aplicaciones que utilicen la representación construida por el sistema. Una de ellas, y que además hemos implementado como parte de este proyecto es un buscador de informes médicos. Para lograr desarrollarlo hemos debido considerar cuál sería la herramienta más adecuada para lograr la indexación de los documentos y realizar consultas sobre ellos. Dado que conocíamos ya la herramienta librería Lucene y dado su gran potencial, no hemos considerado otras herramientas, por lo que a continuación pasamos directamente a describir su funcionalidad y sus principales características.

[20] http://metamap.nlm.nih.gov/MM09_Release_Notes.shtml

2.9.1. *Lucene*

Lucene[21] es una librería para *Java*, desarrollada por Doug Cutting, y orientada para la realización de búsquedas de manera eficaz, en poco tiempo y con una sintaxis muy variada [35]. Nos permite realizar un indexado sobre cualquier tipo de documentos cuyo texto pueda ser extraído de manera que luego se puedan ejecutar consultas y obtener resultados gracias a ellas. Es por esto que es comúnmente utilizado para realizar búsquedas en gran cantidad de páginas web[22], entre las que se encuentran algunas tan conocidas como Twitter, IBM, Apple, Wikipedia o LinkedIn.

En la actualidad se encuentra en la versión 4.3 y se distribuye bajo la licencia *Apache Software License*[23].

Cabe destacar que además existen versiones para otros lenguajes de programación como *Delphi*, *Perl*, *C#*, *C++* , *Python*, *Ruby* y *PHP*.

2.10. Conferencias relacionadas

2.10.1. *TREC*

La conferencia sobre recuperación de textos o *Text REtrieval Conference (TREC)*[24] se inició en 1992 con el fin de servir de apoyo a la investigación dentro del campo de recuperación de información. Nació con la meta de proporcionar aquellos elementos que pudiesen resultar necesarios para la evaluación a gran escala de los distintos métodos de recuperación de información existentes. Está co-patrocinado por el *National Institute of Standards and Technology (NIST)*, y el Departamento de Defensa de EE.UU. y desde la propia página oficial afirman que su intención es:

- fomentar la investigación en el campo de la recuperación de información sobre conjuntos de gran tamaño;
- aumentar la comunicación entre la industria, el mundo académico y el gobierno mediante la creación de un foro abierto que permita el intercambio de ideas;
- acelerar la transferencia de tecnología entre los laboratorios de investigación y los productos comerciales, demostrando para ello la existencia mejoras para problemas del mundo real;
- aumentar por un lado la disponibilidad de técnicas de evaluación que puedan ser utilizadas por la industria y el mundo académico, y por otro el desarrollo de nuevas técnicas que resulten más aplicables a los sistemas actuales.

Para cada *TREC*, se proporciona desde el *NIST* un conjunto de documentos y consultas. El objetivo es que los participantes ejecuten sus propios sistemas de recuperación sobre los

[21] http://lucene.apache.org/
[22] http://wiki.apache.org/lucene-java/PoweredBy
[23] http://www.apache.org/licenses/LICENSE-2.0.html
[24] http://trec.nist.gov/

documentos proporcionados y envíen al *NIST* los resultados para que puedan ser evaluados. Finalmente se termina con un foro en el que los participantes pueden compartir sus experiencias.

Las colecciones de prueba están disponibles para la comunidad que investiga este campo en general, y es por esto último por lo que surge nuestro interés por esta conferencia, ya que para implementar y probar el sistema que hemos desarrollado, se ha utilizado uno de los conjuntos de prueba que estaba compuesto por informes médicos en inglés.

Para poner fin a este apartado podemos comentar que, contando con gran cantidad de participantes, han logrado su objetivo habiéndose comprobado que, durante los primeros seis años de la *TREC*, los métodos de recuperación duplicaron su eficacia. Además, cabe destacar, que han patrocinado las primeras evaluaciones de recuperación de información en lenguajes distintos al inglés, concretamente español y chino dos de los idiomas más hablados en el mundo actualmente.

2.10.2. *CLEF*

La iniciativa *Conference and Labs of the Evaluation Forum* (*CLEF*)[25], antes conocido como *Cross-Language Evaluation Forum*, es una organización cuyo principal objetivo la investigación, innovación, y búsqueda de la información en sistemas multi-lenguaje y con varios niveles de estructuración.

Desde el año 2009, el *CLEF* organiza un conjunto de concursos/conferencias donde se proponen una serie de tareas que los equipos de investigación deberán resolver. En los siguientes apartados comenzaremos comentando las diferentes ediciones que han tenido lugar, para más adelante en el la sección 4.3, pasar a hablar de nuestra propia experiencia y participación.

2.10.2.1. Ediciones

En las siguientes sub-secciones, procederemos a enumerar cada uno de los apartados con que han contado las diferentes ediciones del *CLEF*, así como el lugar y fechas en que se han llevado a cabo. De esta manera, esperamos que el lector pueda entender con más facilidad en qué consiste, y su relación con las tareas de recuperación y extracción de información.

Edición 2009[26]

Esta edición fue desarrollada en Corfu (Grecia) durante los días 30 de Septiembre y 1 y 2 de Octubre.

Las tareas durante este año fueron las siguientes:

- *Multilingual Document Retrieval*

[25] http://www.clef-initiative.eu//
[26] http://www.clef-initiative.eu/edition/clef2009

- *Interactive Cross-Language Retrieval*
- *Multiple Language Question Answering*
- *Cross-Language Retrieval in Image Collections*
- *Multilingual Information Filtering*
- *Cross-LanguageVideo Retrieval*
- *Intellectual Property*
- *Log File Analysis*
- *Grid Experiments*
- *Miscellaneous*
- *MorphoChallenge*

Edición 2010[27]

Esta edición fue desarrollada en Padua (Italia) durante los días 20, 21, 22 y 23 de Septiembre.

Las tareas durante este año fueron las siguientes:

- *Retrieval in the Intellectual Property Domain*
- *Image Retrieval in CLEF*
- *A benchmarking activity on plagiarism detection*
- *Multiple Language Question Answering 2010*
- *Question Answering for Machine Reading Evaluation*
- *Searching Information about Entities in the Web*
- *Cross-lingual Expert Search*
- *Log File Analysis*

Edición 2011[28]

Esta edición fue desarrollada en Amsterdam (Holanda) durante los días 19, 20, 21 y 22 de Septiembre.

Las tareas durante este año fueron las siguientes:

- *Cultural Heritage in CLEF*

[27] http://www.clef-initiative.eu/edition/clef2010
[28] http://www.clef-initiative.eu/edition/clef2011

- *Retrieval in the Intellectual Property Domain*
- *Image Retrieval in CLEF*
- *Uncovering plagiarism, authorship and social software misuse*
- *Question Answering for Machine Reading Evaluation*
- *Log File Analysis*
- *Music Information Retrieval*

Edición 2012[29]

La edición del año 2012 fue desarrollada en Roma, Italia durante los días 17,18,19,20 de Septiembre.

Las tareas del año 2012 fueron las siguientes.

- *Cultural Heritage in CLEF*
- *Retrieval in the Intellectual Property Domain*
- *Image Retrieval in CLEF*
- *Evaluation of XML retrieval*
- *Uncovering plagiarism, authorship and social software misuse*
- *Question Answering for Machine Reading Evaluation*
- *Evaluation of Online Reputation Management Systems*
- *Cross-Language Evaluation for eHealth Document Analysis*

2.11. Punto de partida y conclusiones

Como se ha mencionado en el capítulo 1, partimos de un proyecto encargado de la corrección ortográfica, expansión de acrónimos, detección de las negaciones y extracción de conceptos en informes médicos en español. Esto supone gran parte del procesamiento que es necesario llevar a cabo sobre el informe médico antes de obtener su representación. A pesar de ello, dicho trabajo, como acabamos de mencionar, está realizado para el tratamiento de informes médicos en español, por lo que a pesar de que existen ciertos puntos en común, también existen notables diferencias en cuanto los recursos utilizados, ya que nuestro propósito es que el sistema funcione tanto en inglés como en español y que además permita detectar especulaciones.

[29] http://clef2012.org/

Corrección ortográfica

Para llevar a cabo la corrección ortográfica, el proyecto "Procesador automático de informes médicos" hace uso de Hunspell, aprovechando a su vez de ciertas utilidades del proyecto "AutoIndexer"[30], desarrollado por Díaz et al. [11] del grupo NIL[31] de la Universidad Complutense de Madrid.

Además a la hora de puntuar sugerencias, utilizaron una combinación de diferentes métodos. Por un lado tuvieron en cuenta la distancia de Levenshtein (véase 2.5.2.2) existente entre la palabra detectada como incorrecta y las sugerencias propuestas por Hunspell. Por otro, consideraron además la distancia de teclado comentada en el apartado 2.5.2.4, y por último decidieron que también sería conveniente comparar el código obtenido mediante la aplicación de un algoritmo fonético (2.5.2.5), concretamente una versión propia de *Metaphone* adaptada para el español. Para realizar esta última comparación, el proceso que se llevaba a cabo, era calcular la representación de ambas cadenas y tras ello obtener la distancia de Levenshtein existente entre ellas.

Siguiendo sus pasos, las diferencias son mínimas; cuando se trata de extraer la información de informes en español, utilizamos los mismos métodos mientras que para tratar con informes inglés hemos realizado ligeras modificaciones. En primer lugar, el diccionario a usar ha sido modificado convenientemente para usar la versión inglesa, pero además han sido añadidos diversos términos obtenidos de varios diccionarios médicos pensados para Hunspell[32][33]. En cuanto a la puntuación de sugerencias, puesto que los dos primeros algoritmos son independientes del idioma únicamente hemos modificado la parte relativa al algoritmo fonético, decidiéndonos por *Double Metaphone* y su versión para *Java* proporcionada como parte de *Apache Commons Codec*[34].

Expansión de acrónimos

En cuanto a la expansión de acrónimos, el proyecto del que partimos se basa en un lexicón especializado en español y además posee un mecanismo de desambiguación que cuenta con un sistema de reglas y un algoritmo de aprendizaje automático. Es por ello que se decidió de nuevo no modificar esta parte.

Sin embargo para la expansión de acrónimos presentes en informes en inglés nos decidimos por utilizar MetaMap y el archivo *UDA* que contendrá un conjunto de acrónimos obtenidos a partir de un lexicón en inglés especializado en acrónimos y abreviaturas médicas.

Detección de negaciones

Para lograr la detección de negaciones los desarrolladores del sistema en el que nos basamos decidieron usar una modificación del algoritmo NegEx con elementos distintivos en español, dado que esto nos pareció lo más razonable conservamos, de nuevo, esta parte

[30] http://nil.fdi.ucm.es/index.php?q=node/471
[31] http://nil.fdi.ucm.es/
[32] http://www.e-medtools.com/Hunspel_openmedspel.html
[33] http://mtherald.com/free-medical-spell-checker-for-microsoft-word-custom-dictionary/
[34] http://commons.apache.org/proper/commons-codec/

intacta. En cuando a la detección de negaciones en inglés, inicialmente nos plantemos el uso también de NegEx, pero finalmente dado que MetaMap contaba con una versión mejorada del mismo algoritmo, utilizamos los resultados devueltos por MetaMap.

Detección de especulaciones

Dado que esta característica no estaba disponible en el proyecto "Procesados Automático de Informes Médicos" ha sido necesario implementarla para ambos idiomas. Así decidimos añadir un sistema de detección de frases especuladas basándonos en una variación propia del algoritmo NegEx que tomase como base las ideas ya comentadas anteriormente de Farkas et al. [12].

Detección de conceptos médicos

Para la detección de conceptos en español el proyecto del que partimos hacía uso de *SNOMED-CT*, sin embargo nosotros decidimos aprovechar las capacidades de MetaMap, que nos permite, como ya hemos comentado detectar conceptos médicos utilizando *UMLS* (que a su vez contiene a *SNOMED-CT*), y utilizar esta herramienta. Además gracias a MetamorphoSys hemos podido restringir las fuentes a utilizar y el idioma de las mismas en función del lenguaje en que estén escritos los informes médicos a tratar.

Capítulo 3

Sistema de procesamiento de informes

3.1. Introducción

En este capítulo se presentarán los distintos módulos de que componen el proyecto. Su objetivo es procesar un informe médico corrigiendo las faltas ortográficas que pudiesen hallarse en él, expandiendo sus acrónimos, y detectando negaciones y especulaciones de conceptos médicos presentes en la ontología elegida. Para ello son necesarias varias etapas, y según estas, hemos divido el capítulo en las secciones que se describen a continuación (no incluyendo la actual introducción):

- **Pre-procesamiento.** Los informes que nuestro sistema es capaz de tratar, deben tener una forma determinada para poder ser procesados. A los archivos que poseen dicha estructura, se les ha asignado el formato `.mxml`. Sin embargo, puesto que el proyecto pretende ser lo más genérico posible y hacer que su funcionalidad pueda ser aprovechada independientemente de esta estructura, se ha incluido un módulo en el sistema, al que se ha denominado pre-procesamiento, que permite generar un informe médico con el formato `.mxml` requerido a partir de otro que viene dado como un archivo de texto `.txt` o de un archivo `.xml`.

- **Procesamiento.** Tras esto se pasará a hablar del procesamiento en sí mismo y de sus distintas fases, así como de la representación obtenida a partir de esta extracción de información. El sistema está implementado de manera que pueda ejecutarse sobre informes médicos en inglés o en español, de manera que esta sección a sido a su vez dividida en otras dos que harán referencia al funcionamiento concreto para cada uno de los idiomas.

Para servir como apoyo a las explicaciones que siguen, puede consultarse la figura 3.1, que detalla el funcionamiento del sistema.

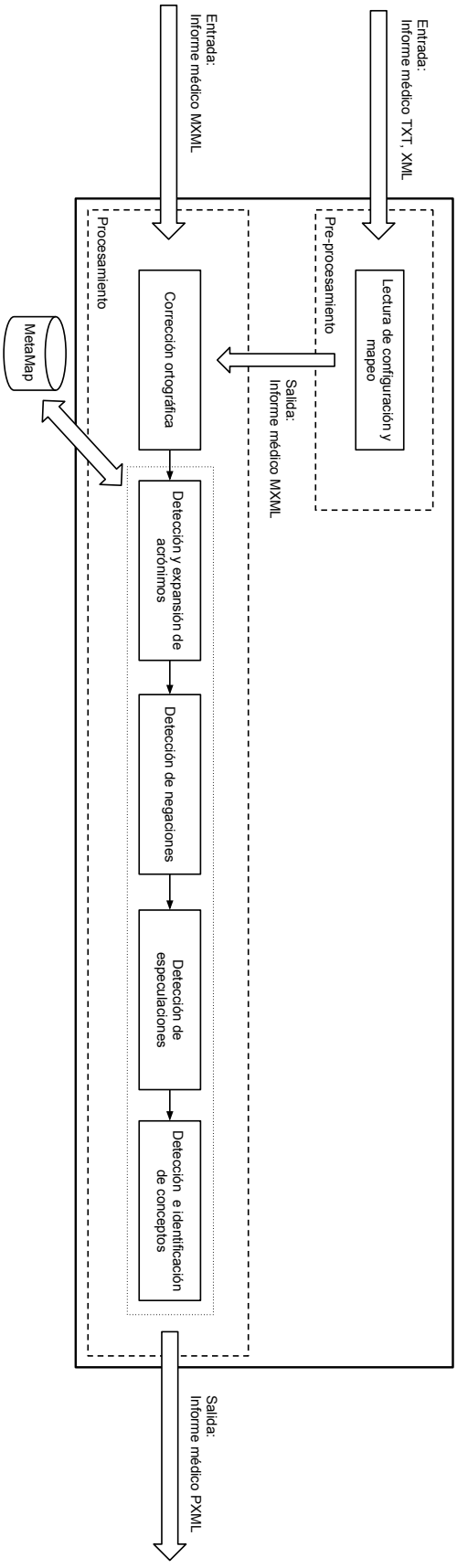


Figura 3.1: Funcionamiento del sistema

3.2. Pre-procesamiento

Esta primera fase, tal como hemos comentado, se encarga de la transformación de un informe en formato `.txt` o `.xml` en otro que posee una estructura determinada y que tiene un formato al que hemos denominado `.mxml`. Comenzaremos pues presentando dicha estructura, para luego detallar la forma en que se obtiene.

3.2.1. Estructura de los informes médicos

El procesador de informes médicos requiere que estos tengan una estructura concreta que ha sido determinada por nosotros mismos atendiendo a los campos que hemos considerado más relevantes a la hora de definir un caso médico. Estos campos son los que se describen a continuación.

- **Sexo, edad:** características que definen a un paciente.
- **Fecha:** fecha en la que ocurrieron los hechos del informe.
- **Motivo de ingreso:** motivo por el cual fue ingresado el paciente.
- **Alergias:** alergias que presenta el paciente.
- **Medicación actual:** medicación previa tomada por el paciente.
- **Antecedentes familiares:** antecedentes familiares del paciente.
- **Antecedentes personales:** antecedentes personales del paciente.
- **Anamnesis:** síntomas que describe el paciente.
- **Exploración:** datos de la exploración realizada por el facultativo.
- **Pruebas complementarias:** pruebas realizadas al paciente y sus resultados.
- **Juicio clínico:** juicio clínico realizado por el facultativo, a su vez se divide en:
  - **Juicio clínico principal**
  - **Juicio clínico secundario 1**
  - **Juicio clínico secundario 2**
- **Plan terapéutico:** plan a seguir indicado por el facultativo.
- **Evolución:** evolución presentada por el paciente.
- **Tratamiento:** tratamiento indicado por el facultativo.
- **Intervención quirúrgica:** intervención o intervenciones realizadas al paciente en caso de que hayan sido necesarias.

3.2.2. Obtención de la entrada con la estructura requerida

Cada hospital puede tener su propio sistema de informes médicos y es por ello que, como se mencionó anteriormente, se ha desarrollado una herramienta que transforma cualquier tipo de informe médico que esté en formato `.xml`, en un archivo `.mxml` que tenga la estructura adecuada para poder ser tratado por nuestra aplicación. Esta transformación se realiza en base a un archivo de configuración que debe indicar cómo debe realizarse el mapeo entre las diferentes secciones de ambos archivos, es decir que tiene que especificar a qué sección del `.mxml` requerido corresponde cada una de las presentes en el archivo `.xml` original.

Además puede darse la situación de que los informes poseídos sean documentos con formato `.txt`, es decir archivos en que no haya ninguna separación por secciones. Este caso también se ha tenido en cuenta y a causa de ello se ha otorgado a la aplicación la posibilidad de leer un informe y obtener el `.mxml` requerido mediante la detección de conjuntos de palabras al inicio de cada frase, puesto que aunque las distintas secciones del texto no estén explícitamente separadas hemos comprobado que siempre empiezan con un título que define la sección. Para que pueda ser configurable dependiendo de cuáles sean las palabras por las que comienzan las secciones, se ha creado un archivo de configuración en el que se puede indicar por qué palabra o conjunto de palabras suele comenzar cada una de ellas. Además hay otros datos como la edad y el sexo del paciente, o la fecha del informe que no están como secciones sino dentro del texto, estos datos se buscan mediante la búsqueda de expresiones regulares.

Los ficheros de configuración podrán modificarse desde la interfaz gráfica.

3.3. Procesamiento

Para que sea posible obtener la representación de un informe médico es necesario realizar previamente otras fases de procesamiento. Estas fases son: corrección ortográfica, expansión de acrónimos, identificación de negaciones, identificación de especulaciones y extracción de conceptos.

Una vez procesado el informe procederemos a guardarlo como un archivo `.pxml`, este tipo de archivo tendrá los mismos campos que el archivo `.mxml` pero en lugar de tener texto plano, tendrá los conceptos extraídos en el apartado anterior y el estado de estos (ver listado de código 3.1).

```
<contenido id="rutaArchivo.pxml">
   <sexo></sexo>
   <edad></edad>
   <fecha></fecha>
   <!-- Secciones de informe -->
   <motivo_ingreso>
      <!-- id es el CUI del concepto -->
      <!-- nombre la cadena de texto que lo representa -->
      <!-- estado puede ser AFIRMADO, NEGADO o ESPECULADO -->
      <concepto id="" nombre="" estado=""/>
      ...
      <concepto id="" nombre="" estado=""/>
```

```
13      </motivo_ingreso>
14      ...
15      <plan_terapéutico>
16          <concepto id="" nombre="" estado=""/>
17          ...
18          <concepto id="" nombre="" estado=""/>
19      </plan_terapéutico>
20  </contenido>
```

Listado de código 3.1: Estructura de los archivos `.pxml`

Los primeros campos del informe (`sexo`, `edad` y `fecha`) no sufren cambios dado contienen datos muy concretos que no necesitan ser tratados.

En el resto de campos del informe se sustituirá el texto plano por los conceptos que contuviese dicho texto y el estado de cada uno de cada uno de ellos. Este estado podrá ser `AFIRMADO`, `NEGADO` o `ESPECULADO`.

La representación de los informes es una parte muy importante de nuestra aplicación ya que gracias a ella seremos capaces de tratar un informe médico de una forma mucho más sencilla que la forma en que deberíamos hacerlo si tratásemos con el texto original, además a partir de esta representación pueden obtenerse diversas aplicaciones. Un ejemplo claro, es un buscador, aplicación desarrollada también como parte de este proyecto.

A continuación se detallará en que consiste cada una de las fases que mencionábamos anteriormente, tanto para inglés como para español.

3.3.1. Procesamiento de informes médicos en inglés

A lo largo de esta subsección describiremos brevemente el funcionamiento de cada una de las fases que son llevadas a cabo para lograr obtener, a partir de un informe médico en inglés, con formato `.mxml`, su representación como archivo `.pxml`.

3.3.1.1. Corrección ortográfica

Para completar esta fase se ha hecho uso de Hunspell y de una de las versiones inglesas del diccionario[1] ampliado con gran cantidad de términos relacionados con la medicina. Para realizar la corrección automática, se llevan a cabo los pasos que se describen a continuación, y que son los mismos que se realizaban en el caso del "Procesador automático de informes médicos".

1. En primer lugar se divide el texto a procesar en *tokens* y se busca cada uno de ellos en el diccionario indicado en el archivo de configuración correspondiente (ver apéndice Configuración).

2. En caso de que el *token* se encuentre en el diccionario, consideramos que la palabra está escrita correctamente, pero en caso de que no se encuentre, se identificará como falta ortográfica, encargándose Hunspell de realizar un conjunto de sugerencias.

[1] http://hunspell.sourceforge.net/

3. Para cada término propuesto por Hunspell se asigna una puntuación mediante la combinación de las tres medidas de comparación utilizadas:

   - distancia de Levenshtein;
   - distancia de teclado;
   - distancia de Levenshtein entre los códigos de ambas palabras calculados mediante un algoritmo fonético.

   La puntuación que se asigna a la sugerencia dependerá de su distancia a la palabra escrita incorrectamente, y se moverá en un intervalo $[0,\ 1]$ En concreto:

   - *si* $Distancia < 10$ *entonces* $Puntuación = 1 - {}^{Distancia}/_{10}$
   - *si* $Distancia \geq 10$ *entonces* $Puntuación = 0$

   Para combinar las tres puntuaciones realiza una media ponderada entre los resultados. El peso asignado por defecto a cada método es el mismo, pero para modificarlo puede editarse el archivo de configuración asociado.

4. En caso de que alguna de las sugerencias obtenga una puntuación superior a un umbral determinado (que de nuevo se encuentra en un archivo de configuración), se tomará como la opción acertada y se sustituirá automáticamente. En caso de que haya varias sugerencias con la misma puntuación, se escogerá la primera de ellas.

Finalmente cabe recordar que, dado que partíamos de un proyecto que trataba con informes en español, a la hora de hacer la elección del algoritmo fonético escogimos *Double Metaphone* tal como se comentó en el apartado 2.11.

3.3.1.2. Expansión de acrónimos

Para la expansión de acrónimos se ha utilizado MetaMap, esta herramienta tiene diferentes formas de expandir acrónimos y hemos utilizado dos de ellas.

- Acrónimos auto-anotados: Son aquellos acrónimos que el redactor del informe escribe en algún momento junto a su expansión correcta (normalmente es en la primera aparición). MetaMap supone que a partir de ese momento, si vuelve a aparece un acrónimo igual, realizará la expansión a aquello que esté indicado entre los paréntesis. Esto es muy utilizado en medicina, puesto que el mismo acrónimo tiene diferente significado dependiendo de la rama médica en la que se este desarrollando el informe.
- Archivo UDA: MetaMap desde su versión del año 2009 incluyó la posibilidad de lanzar los servidores con un archivo llamado UDA[2]. Nosotros hemos creado un archivo `.txt` en el que hemos añadido acrónimos en inglés y su expansión, MetaMap cada vez que encuentre uno de estos acrónimos lo expandirá por la expansión indicada. Para crear el archivo txt hemos incluido los acrónimos que hemos encontrado en diversas páginas web con listas de acrónimos y sus expansiones[3][4][5].

[2] http://metamap.nlm.nih.gov/MM_2011_ReleaseNotes.pdf#page=9
[3] http://www.acronymslist.com/cat/medical-acronyms.html
[4] http://www.medindia.net/acronym/index.asp
[5] http://www.globalrph.com/abbrev.htm

3.3.1.3. Identificación de negaciones

Para la detección de las negaciones se utilizan los resultados obtenidos durante el procesado con la herramienta de MetaMap.

En esta herramienta, se le facilita una frase y mediante el uso de una API en *Java*, se permite obtener la siguiente información sobre una frase negada:

- Lugar donde se ha detectado una frase negada en un conjunto de palabras.
- Palabra que hace que la frase esté negada.
- Listado de conceptos que están negados.

Tomando esta información, generamos una `AnotacionNegada` que contiene toda esta información y que posteriormente se utilizará para producir la salida del procesado de un informe.

3.3.1.4. Identificación de especulaciones

Como se indicó en el capitulo previo, la identificación de especulaciones se basa en una modificación del algoritmo NegEx para la detección de la negación para que sea posible detectar frases especuladas o condicionadas en función de ciertas palabras claves.

La detección se realiza de la siguiente manera:

- Si se trata de un concepto que es un adverbio o un adjetivo con un cierto significado especulativo, la frase se extiende desde ese adverbio/adjetivo hasta el final de la misma.
  - Los adjetivos considerados son: *probable, likely, possible, unsure, often, possibly, allegedly, apparently* y *perhaps*.
  - Los adverbios considerados son: *widely, traditionally ,generally, broadlyaccepted, widespread, global, superior, excellent, immensely, legendary, best, largest, most, prominent, clearly, obviously* y *arguably*.
- Si se trata de un verbo con significado especulativo o un modificador de verbo que añade el sentido especulativo, la frase se extiende hasta el final o la primera conjunción que se detecte, en este ultimo caso, las dos frases unidas por esta conjunción se considerarán especulativas.
  - Los verbos considerados son: *suggest, question, presume, suspect, indicate, suppose, seem, appear, favor, there is* y *there are*.
  - Los modificadores de los verbos considerados son: *may, might, would* y *should*.
  - Las conjunciones consideradas son: *or*, *and* y *either*.

3.3.1.5. Extracción de conceptos

El proceso de extracción de conceptos es muy complicado, por una razón muy simple, necesitamos contextualizar cada una de las palabras presentes en el texto para poder determinar correctamente que palabra es y si esta asociado a un concepto, y dado el caso, determinar a que concepto esta asociado.

El proceso de la detección de un concepto, funcionalmente es muy simple, podríamos ir recorriendo cada *token* presente en una frase, y si el token que estamos analizando actualmente pertenece a un conjunto de *tokens* que tienen cierto significado médico, determinaríamos que la detección ha sido correcta y lo anotaríamos para luego trabajar con él.

Pero nos encontramos con varios problemas en este planteamiento inicial:

1. Diferentes palabras pueden referirse a un mismo concepto, como pueden ser: "Ataque al corazón" e "Infarto de miocardio".
2. La polisemia es muy compleja en el análisis del lenguaje natural, dándose casos en los que los diferentes significados de la misma palabra son aceptados en un ámbito médico, como puede ser cabeza, dado que puede referirse a la cabeza de una persona o a la cabeza de un clavo quirúrgico o a la cabeza (extremo) de un hueso.

Procesador Automático de Informes Médicos

En el trabajo del año previo al nuestro, y en que que nos basamos, lo que hacían durante esta fase era simple pero tenía con cierta efectividad. Poseían una base de datos en la que estaban todos los conceptos de *SNOMED-CT* en español así los nombres de las partes del documento en que podía aparecer el citado concepto.

El problema que conllevaba el hecho de usar esta técnica (obviando el hecho de que estuviese en español), era el apartado de la desambiguación. Para solucionar este problema, en su caso los desarrolladores determinaron qué tipo de conceptos podían aparecer en un apartado concreto.

Aunque este método devolvía unos resultados muy buenos para los informes de los que disponían, si intentamos extrapolar estos resultados a otro tipo de informe, o uno que no esté relacionado en absoluto con ellos, se pierde efectividad de una manera destacada.

MetaMap

Teniendo en cuenta estos errores, inicialmente se planteó como opción, tomar las bases de datos con que contábamos y completarlas en inglés y utilizar las reglas de desambiguación que poseíamos. Inmediatamente nos dimos cuenta que esta última decisión suponía un fallo muy grave, ya que la gramática de ambos idiomas es muy diferente y la desambiguación no se efectuaría con éxito.

Posteriormente, se determinó que la detección de conceptos era mejor ejecutarla con la ayuda de MetaMap, la cual permite, como ya comentamos anteriormente, la detección de los conceptos contenidos en diferentes bases de datos.

La utilización de MetaMap, también suponía un problema que debíamos solventar inmediatamente, y es que se obtenían demasiados resultados para ciertos conjuntos de palabras, por ejemplo para expresión *Lung cancer* (cáncer de pulmón), se recuperaban los resultados *Lung* (pulmón)*, cancer* (enfermedad) y *Lung Cancer* (enfermedad concreta que afecta a los pulmones).

La primera aproximación para evitar todo este ruido, o al menos minimizarlo, fue desarrollar una ontología de aparición, de manera que indicábamos para cada zona de un informe qué tipo de conceptos podían aparecer en ella, pero nos encontramos con el problema de determinar precisamente eso, puesto que no somos expertos en este campo y no contábamos con la colaboración de ninguno como para determinar si un tipo debía o no ser válido en función de la sección del informe en que estuviésemos situados. Además no poseíamos una gran cantidad de informes médicos, por lo que tampoco nos resultaba posible desarrollar un algoritmo de aprendizaje.

La segunda aproximación fue la utilización del desambiguador que ofrece MetaMap, y que asigna una puntuación a cada concepto. De esta manera volviendo al caso anterior, a pesar de que se obtienen los 3 resultados, debido al contexto, *Lung Cancer* posee mayor puntuación que el resto y por tanto es más probable que sea el adecuado.

Aún con el desambiguador, continuábamos obteniendo demasiado ruido, y tal como señalábamos en el apartado , vimos que venía provocado por el uso de todas las bases de datos de las cuales dispone MetaMap (un total de 112 en inglés[6]). Por este motivo, se planteó una tercera aproximación que consistía en el desarrollo de nuestra propia base de datos, a partir de las ya existentes y eliminando todas las que no aportaban la información que nosotros buscábamos.

Aunque MetaMap, atendiendo a la resolución de este problema, facilita información que ayude a crearlas correctamente, cabe destacar que dicha información es incompleta por lo que tras ponernos en contacto con los responsables[7][8], logramos desarrollar de manera satisfactoria las bases de datos que utilizamos. Los pasos seguidos para ello, se encuentran detallados en el apéndice Creación de una ontología personalizada.

Con todo esto, finalmente detectamos los conceptos y nos quedamos solo con el que tiene mayor puntuación. En caso de que se recuperen varios resultados con la misma puntuación se tomará por bueno aquel que primero que aparece, almacenándose junto con la siguiente información:

- Lugar de aparición
- Extensión del concepto (debido a que pueden ser mas de una palabra).
- La puntuación
- Aquellas palabras por las que es preferible que sea llamado (por ejemplo, para *infarto de miocardio* y para *ataque al corazón*, las palabras preferidas son *infarto de miocardio*).
- Estado en el que está el concepto, es decir, si está afirmado, negado o especulado.

[6] http://www.nlm.nih.gov/research/umls/sourcereleasedocs/index.html#
[7] wjrogers@mail.nih.gov
[8] flang@mail.nih.gov

3.3.2. Procesamiento de informes médicos en español

3.3.2.1. Corrección ortográfica

La fase de corrección ortográfica automática llevada a cabo durante el procesamiento de un informe en español no se ha variado en absoluto respecto a descrito por Barahona et al. [5]. Los pasos que componen esta parte del tratamiento de informes médicos en español, es equivalente a la detallada en el apartado de este mismo capítulo. Si bien es cierto que varía en los siguientes aspectos:

- A la hora de buscar un *token* para comprobar su validez ortográfica, éste se busca, como es de suponer, en un diccionario con términos en español que fue mejorado por los autores anteriormente citados, con conceptos y acrónimos médicos que permitan evitar falsos positivos.
- El algoritmo fonético que determina cuál es el código asociado a una palabra, no es *Double Metaphone* como lo era para inglés. Suponemos que habría sido útil hacer uso de la versión para español de *Metaphone 3*, presentado en la subsección 2.5.2.5, sin embargo este algoritmo es comercial, por lo que se decidió que la mejor y más simple opción sería mantener esta parte del código intacta en relación al "Procesador automático de informes médicos" y por tanto se conservó la adaptación de *Metaphone* original para español.

  Las reglas por las que se rige esta adaptación son las siguientes:

  - C = Z, en caso de que vayan seguidas de E o I.
  - C = K, cuando van seguidas de A, O, U o cualquier otra consonante, excepto la C y la H.
  - G = J, en caso de que vayan seguidas de E o I.
  - Ll = Y.
  - B = V.
  - U = W.
  - X = S, cuando se trata de la primera letra de la palabra.
  - QU = K.
  - Q = K, cuando van seguidas de cualquier letra, excepto la U.
  - La H se omite por considerarse "muda".

3.3.2.2. Expansión de acrónimos

Para la expansión de acrónimos en español se utiliza el mismo sistema que en el proyecto en que nos basamos.

Para detectar los acrónimos se divide el texto en palabras, cada palabra se busca en el lexicón de acrónimos y si se encuentra es clasificada como acrónimo. El lexicón de acrónimos tiene más de 3000 acrónimos y para obtenerlo se ha utilizado un lexicón del ámbito médico presente en el proyecto AutoIndexer [11], completado con un diccionario de siglas médicas del Ministerio de Sanidad y Consumo.

Para la desambiguación de acrónimos se ha utilizado un sistema de reglas y un sistema de aprendizaje que deciden que expansión corresponde con el acrónimo en función del contexto en el que esté el acrónimo y la sección del informe en la que aparezca.

3.3.2.3. Identificación de negaciones

Igual que para la detección de las negaciones en inglés, se utiliza un algoritmo NegEx para la detección de la negación, solo que en esta ocasión, se utiliza el algoritmo implementado directamente y no la versión incluida como parte de MetaMap. Esto se debe, a que dado que partíamos de un proyecto que ya realizaba esta fase de manera razonablemente buena, no se consideró prioritario modificarlo.

Esta decisión de diseño se tomó por una razón muy simple, el algoritmo NegEx incluido en el núcleo de MetaMap está adaptado para la utilización del mismo en inglés y no en español, por lo que el uso de este algoritmo en un informe en español no produciría los resultados que cabría esperar.

El algoritmo aplicado, pues, funciona de la siguiente manera:

- Disponemos de un conjunto de palabras claves clasificadas en 4 subconjuntos: Negaciones, PostNegaciones, Conjunciones y PseudoNegaciones.
  - **Negaciones**: son las palabras tales que todo lo que sigue después de ellas y hasta el final de la frase está negado.
  - **PostNegaciones**: son el conjunto de palabras tales que los conceptos que hay desde el comienzo de la frase hasta esta palabra están negados.
  - **Conjunciones**: unen diferentes frases negadas, tal que si se ha detectado una negación y a continuación hay una conjunción, la frase que sigue también estará negada.
  - **PseudoNegaciones**: es el conjunto de palabras tales que parecen una negación pero en realidad no lo son, por lo que cuando se detecta un miembro de este grupo, se ignora y la frase continuará afirmada salvo que se detecte lo contrario por cualquiera de los otros 3 grupos.

1. Conforme vamos avanzando por el texto leyendo uno a uno los *tokens* del que esta compuesto, si detectamos alguno perteneciente a uno de los conjuntos previamente citados, se procederá como se indica para cada conjunto, en cualquier otro caso, se leerá el siguiente *token*.
2. Una vez que se ha determinado si una frase está afirmada o negada, se continúa con el resto de las frases volviendo al paso 1.

3.3.2.4. Identificación de especulaciones

El funcionamiento de la detección de las especulaciones en español es el mismo que para la detección en ingles, solo que cambiando las palabras claves.

- Adjetivos: “probable”, “posible”, “inseguro”, “a menudo”, “probablemente”, “pretendidamente”, “aparentemente” y “tal vez”.
- Adverbios: “extensamente”, “tradicionalmente”, “en general”, “extendido”, “global”, “superior”, “excelente”, “inmensamente”, “legendario”, “mejor”, “más grande”, “más prominente”, “claramente”, “obviamente” y “podría decirse que”
- Verbos: “sugerir”, “cuestión”, “presumir”, “sospechar”, “indicar”, “suponer”, “parecer”, “aparecer”, “favorecer” y “hay”.
- Modificadores de Verbos: No hay ninguno al no existir ese tipo gramatical en el castellano.
- Conjunciones*:* “y”*,* o y “cualquiera de los dos”.

3.3.2.5. Extracción de conceptos

Para la extracción de conceptos en español, tras pasar por los mismos pasos que para el ingles, se desarrollo otra base de datos que se puede leer desde MetaMap, con la característica de que esta base de datos está desarrollada con las bases de datos en español disponibles en MetaMap, siendo estas[9] :

- *Physicians’ Current Procedural Terminology, Spanish Translation, 2001*
- *ICPC, Spanish Translation, 1993*
- *Medical Dictionary for Regulatory Activities Terminology (MedDRA), Spanish Edition, 15.0*
- *Descritores en Ciencias de la Salud [Spanish translation of the Medical Subject Headings], 2012*
- *SNOMED Clinical Terms, Spanish Language Edition, 2012_04_30*
- *WHOART, Spanish Translation, 1997*

El proceso de desambiguación en este caso también se utiliza lo proporcionado por MetaMap, debido a que durante la construcción de las bases de datos, se generan los archivos que luego también usará el desambiguador, por lo que los resultados serán mejores que usando la información del año anterior.

La información utilizada en esta fase es exactamente la misma que en la fase en inglés.

[9]Datos tomados de la herramienta de MetamorphoSys

Capítulo 4

Usos y aplicaciones

4.1. Introducción

Este capítulo pretende demostrar el interés actual que existe en las tareas de recuperación y extracción de información y más concretamente en lo relativo al ámbito de la medicina, en que se encuadra nuestro trabajo, además de presentar las posibles aplicaciones que puede tener el proyecto que hemos desarrollado. Así pues, las secciones que incluye, además de la presente introducción, son las siguientes:

- **Indexación y búsqueda de informes de los informes procesados.** Comenzaremos describiendo algunos de los usos que se podrían dar al sistema que hemos desarrollado, concretamente hablaremos sobre el buscador de informes implementado y sobre otras extensiones que podrían obtenerse a partir de él.
- ***CLEF 2013*.** Tras ello, dedicaremos una sección completa a la edición del *CLEF* de este año. Esto se debe a que supone un buen ejemplo del interés que existe en la realización de este tipo de proyectos, pero especialmente a que, como parte del trabajo llevado a cabo, decidimos presentarnos como participantes, modificando algunas partes de nuestra implementación, con el fin de poder llevar a cabo las tareas que se proponían y poder evaluar así, además, el sistema implementado. Finalmente se recogen y comentar los resultados de dicha evaluación.

4.2. Indexación y búsqueda de informes de los informes procesados

Una vez obtenido el conjunto de archivos `.pxml`, el siguiente paso natural consistía tratarlos para poder sacar provecho de la información que contenían. De esta manera decidimos desarrollar un buscador de informes médicos que no solo buscase por palabras, sino que pudiese "ver" más allá, descartando términos no relevantes, y teniendo en cuenta únicamente los conceptos médicos, además de su estado (afirmado, negado o especulado) y la sección del informe médico en la que se encuentran. Para consultar la estructura de estos archivos, se referencia al lector al listado de código 3.1.

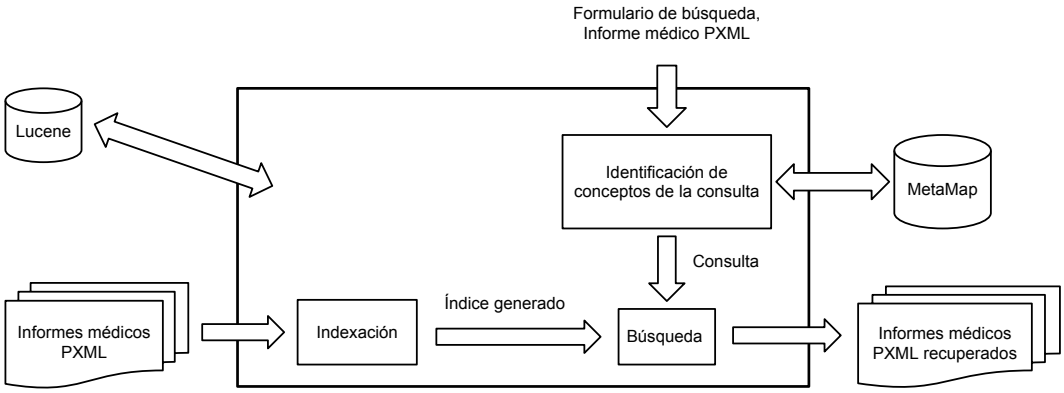


Figura 4.1: Funcionamiento del buscador de informes médicos

De esta manera implementamos dicho buscador, que permitiría dos modos de uso. El primero de ellos, daría al usuario la posibilidad de rellenar un formulario con aquellas palabras que quisiese buscar, indicando el estado y la sección en que desea que se encuentren. El segundo modo de utilización daría la opción al individuo que estuviese ejecutando la aplicación, de seleccionar un informe en formato `.pxml` para intentar localizar aquellos similares a él.

Esta aplicación puede resultar de interés tanto para profesionales como para investigadores, ya que gracias a ella podrían localizar informes médicos que reúnen ciertas características no solo mucho más rápido de lo que podrían hacerlo manualmente, también de manera considerablemente más efectiva que si lo hiciesen sobre los informes que no han sido tratados por el sistema. Esto se debe a que, gracias a este procesamiento, somos capaces de recoger aquello relevante omitiendo lo que no nos interesa, por lo que a la hora de realizar una consulta, el buscador no se verá tan afectado por el posible ruido que pueda introducir aquello que no es importante para la búsqueda.

Por otro lado, el hecho que el buscador incluya la posibilidad de localizar informes similares a uno dado, da la oportunidad, no solo de clasificarlos en grupos de informes similares, sino además de comparar diferentes casos médicos apoyando los diagnósticos dados.

Asimismo la obtención de datos y resultados estadísticos se convierte en una tarea sencilla. Sería factible ampliar el funcionamiento del buscador de modo que se pudiesen generar gráficas y cuadros que reuniesen ciertos datos indicados previamente. Así por ejemplo, sería viable la recopilación de información referente a la distribución de una determinada enfermedad, en hombres y en mujeres, a lo largo de los años, o sobre ciertos rangos de edad.

El proceso que sigue este sistema de búsqueda, está descrito en la figura 4.1. A continuación procedemos a explicar con más detalle tanto las tecnologías utilizadas para llevarlo a cabo como los propios procesos de indexación y búsqueda.

4.2.1. Tecnologías Utilizadas

Para realizar este sistema se utilizó la librería de *Java, Lucene*[1] que permite realizar la indexación información y realizar búsquedas sobre el índice creado de manera eficaz. El inconveniente que presenta esta librería es la gran cantidad de trabajo que realiza durante el proceso de indexación y la enorme consumición de recursos y tiempo que ello conlleva, sin embargo a la hora de realizar una búsqueda esta se lleva a cabo en un periodo muy corto de tiempo, permitiendo además la utilización de consultas con modificadores de tipo *boolean*.

La información se deberá almacenar en una clase de *Lucene* llamada `Document`, que representa un documento y está compuesto por un conjunto de `Fields`, (clase perteneciente igualmente a esta librería).

Los `Fields` se dividen en 2 categorías:

- Aquellos que son utilizados para la indexación de información.
- Aquellos que son utilizados para el filtrado de la información.

Por como hemos determinado que van a ser nuestras búsquedas, se utilizarán únicamente los `Fields` de la primera categoría, lo que permitirá entre otras cosas, buscar informes de un sexo concreto sin tener en cuenta ninguna otra información.

4.2.2. Indexación

Inicialmente se creó un único índice en el que cada documento era de la siguiente manera:

`ID_SNOMED-CT|ID_INFORME|Estado concepto|Sección concepto|SEXO|RAZA|EDAD|FECHA`

Tras documentarnos correctamente sobre el uso de *Lucene*, descubrimos que esta manera de clasificar los datos no era la más adecuada y que en su lugar deberíamos crear un índice independiente para cada combinación de sección del informe y estado posible, por ejemplo `Alergias ESPECULADO`.

Con esta nueva especificación, un documento queda representado de la siguiente manera:

`ID_SNOMED-CT`[2]`|ID_INFORME|SEXO|RAZA|EDAD|FECHA`

dando lugar a un total de

$$16\ secciones * 3\ estados = 48\ índices.$$

[1] http://lucene.apache.org/core/

[2] Podrían ser más de uno: todos aquellos asociados a la sección y estado representados por el índice

4.2.3. Consultar en el Índice generado

A la hora de realizar una búsqueda, se utilizará un método que hemos desarrollado que recibe los siguientes parámetros:

- *String* con la búsqueda formateada
- Campo en el que buscar(Antecedentes_Familiares, por ejemplo)
- Estado en el que buscar(Afirmado, por ejemplo)
- Número de resultados máximos a obtener

4.2.3.1. Formato de las consultas

Siguiendo las directivas de *Lucene*, la búsqueda deberá tener el siguiente formato:

`NombreCampo:(‘‘Palabra Clave’’ Factor_Booleano ‘‘Otra Palabra Clave’’) Factor_Booleano OtroCampo:(‘‘Palabra Clave’’ Factor_Booleano ‘‘Otra Palabra Clave’’)`, siendo muy importantes, los dos puntos y las comillas.

4.3. *CLEF 2013*

4.3.1. Introducción

Ya en la sección 2.10.2 presentamos en qué consiste el *CLEF* y las ediciones de años pasados. Durante el resto de esta sección nos centraremos en la edición del año 2013[3], que se desarrollará en Valencia (España) durante los días 23, 24, 25 y 26 de Septiembre de este mismo año. Dada nuestra participación en ella procedemos a discutir cuáles han sido las tareas propuestas y cuál ha sido nuestro papel en ellas dando un uso más al proyecto elaborado.

Las tareas fijadas para llevar a cabo en esta ocasión han sido las siguientes:

- *CHiC - Cultural Heritage in CLEF*
- *CLEFeHealth - CLEF eHealth Evaluation Lab*
- *CLEF-IP - Retrieval in the Intellectual Property Domain*
- *ImageCLEF - Cross Language Image Annotation and Retrieval*
- *INEX - INitiative for the Evaluation of XML retrieval*
- *PAN - Uncovering Plagiarism, Authorship, and Social Software Misuse*
- *QA4MRE - Question Answering for Machine Reading Evaluation*
- *QALD-3 - Question Answering over Linked Data*

[3] http://www.clef2013.org/index.php

- *RepLab 2013*

Dada la naturaleza del proyecto que estábamos desarrollando, nuestro propio director, Alberto Díaz Esteban, nos informó sobre la existencia del *CLEF 2013* animándonos a participar, concretamente en lo relativo al apartado *CLEF eHealth Evaluation Lab*, ya que precisamente versaba sobre el tratamiento de información contenida en textos relacionados con la salud y la medicina.

Dicho apartado se dividía a su vez en tres tareas independientes cuyo único nexo era que, para la extracción de información, se debía hacer uso de las bases de datos de *UMLS* y concretamente de *SNOMEDCT(2011AA)*.

4.3.2. Tarea 1

La realización de esta tarea consistiría en la detección de conceptos en un informe médico escrito, como es de suponer, en inglés.

A la hora de detectar un concepto, se debería indicar cuál era la posición del carácter con que comenzaba y cuál era la posición del carácter con que terminaba. Además sería necesario especificar igualmente su ID, por lo que, para obtener unos resultados óptimos cobraría suma importancia la existencia de una fase de desambiguación. En caso de que un concepto no se encontrase en la base de datos de *SNOMED-CT*, se tendría devolver el concepto pero no su ID.

Por otro lado la especificación de la tarea señalaba que el desarrollador estaría obligado a recuperar únicamente los conceptos cuyo campo semántico fuese uno de los siguientes:

- *Congenital Abnormality*
- *Acquired Abnormality*
- *Injury or Poisoning*
- *Pathologic Function*
- *Disease or Syndrome*
- *Mental or Behavioral Dysfunction*
- *Cell or Molecular Dysfunction*
- *Experimental Model of Disease*
- *Anatomical Abnormality*
- *Neoplastic Process*
- *Signs and Symptoms*

4.3.2.1. Realización de la Tarea 1

Con el fin de implementar el sistema que permitiese cumplir la especificación anterior, se modificó ligeramente el trabajo obtenido hasta el momento para poder así obtener unos resultados más próximos a los solicitados. A continuación describiremos qué partes y de qué manera fueron modificadas.

- Se eliminaron las siguientes fases del procesamiento ya que se consideró que no serían necesarias:
  - Deshabilitamos el módulo de corrección ortográfica, pues determinamos que los informes no contendrían faltas ortográficas.
  - Deshabilitamos el módulo de expansión de acrónimos, pues esto formaba parte de la tarea 2 y pensamos que introduciría mucho ruido.
  - Deshabilitamos el módulo de detección de las negaciones, pues la tarea consistía en la detección de conceptos, independientemente de si estaban negados o afirmados.
  - Deshabilitamos el módulo de detección de las especulaciones, por las mismas razones que el caso anterior.
- Se modificó la fase de detección de conceptos con el objeto de que se adecuase las siguientes reglas:
  - De entre todos los resultados que devuelve MetaMap, se seleccionarán exclusivamente aquel que posea un mejor *score*.
  - Únicamente se comprobarán los resultados cuyo tipo semántico sea uno de los solicitados.
  - Si el mejor resultado no pertenece a *SNOMED-CT*, se fijará como ID las palabras: *"CUI-Less"*.
- Se eliminó la diferenciación de las posibles secciones que pudiesen existir en los informes, procesando cada uno de ellos como si contase únicamente con una.

  Cabe señalar que se decidió adoptar esta medida debido al poco tiempo disponible para realizar la tarea, y aún a sabiendas de que este cambio en concreto no sería el más apropiado, ya que no estaríamos obteniendo los mejores resultados.
- Se realizaron diversos cambios para que el archivo pudiese ser leído desde un documento de texto plano, en lugar de desde un documento con formato `.xml`.
- Por último se ajustó la salida proporcionada por el sistema para generar un archivo que contase con el formato apropiado y que pudiese ser posteriormente enviado para su evaluación.

4.3.2.2. Resultados obtenidos

A la hora de evaluar nuestro sistema y la calidad de los resultados que proporciona, se toman los datos obtenidos tras procesar un informe y se calculan los cuatro valores

siguientes: *Precision, Recall, F1-score y Accuracy,* que explicaremos más adelante, utilizando para ello unos informes anotados. Estos informes anotados, fueron en nuestro caso, aquellos que nos facilitaron desde la organización del *CLEF*.

Para poder obtener estas cuatro medidas de evaluación, es necesario calcular previamente otras cuatro que detallamos a continuación:

Verdadero Positivo: Son aquellos valores que el sistema detecta como válidos y están en los informes anotados por el humano experto. De ahora en adelante los llamaremos VP.

Verdadero Falso: Son aquellos valores que el sistema detecta como inválidos y debe hacerlo así puesto que no están en los informes anotados. A partir de ahora los llamaremos VN.

Falso Positivo: Son aquellos valores que el sistema detecta como válidos, cuando en realidad se *no* encuentran anotados por el experto en los informes y por tanto debería detectarse como válidos. Desde ahora nos referiremos a ellos como FP.

Falsos Negativos: Son aquellos valores que el sistema detecta como inválidos, cuando en realidad *sí* se aparecen anotados en los informes. A partir de ahora los llamaremos FN.

Adicionalmente podemos definir otras dos medidas:

1. $Correctos = VP + VF$. Valores que nuestro sistema determina como correctos.
2. $Total$. Cantidad total de valores, que dependerá de si tratamos con el modelo estricto o el relajado.

Las cuatro primeras forman lo que se conoce como tabla de contingencia y que podemos visualizar en el cuadro Tabla de contingencia.

		Humano experto	
		Verdadero	Falso
Sistema	Verdadero	VP	FP
	Falso	FN	VN

Tabla 4.1: Tabla de contingencia

Una vez calculados estos valores, las fórmulas de *Precision, Recall, F1-Score y Accuracy* se definen de la siguiente manera:

- *Precision* se define como: $\frac{VP}{VP+FP}$
- *Recall* se define como: $\frac{VP}{VP+FN}$

- *F1-score* se define como: $2 * \frac{Precision * Recall}{Precision + Recall}$
- *Accuracy* se calcula de la siguiente manera: $\frac{Correctos}{Total}$

Estas dos primeras medidas muy relacionadas, de hecho se ha comprobado que mantienen una relación inversa, de manera que a mayor *Precision* menor *Recall* y viceversa [9]. Debido a nuestra participación en el CLEF, se calculó nuestra *Precision, Recall, F1-Score y Accuracy* por parte de la organización con un conjunto de informes de los que desconocíamos su resultado correcto y se ordenaron los resultados de todos los participantes.

A la hora de realizar los cálculos, estos se obtienen de dos maneras distintas, de manera estricta y de manera relajada:

- Estricta: los fallos penalizarán los resultados correctos.
- Relajada: los fallos *no* penalizarán los resultados correctos.

Los valores de *Precision, Recall, F1-Score y Accuracy*, se calcularon de manera estricta y relajada. Además, se presentaron dos resultados distintos, uno utilizando las base de datos de *SNOMED-CT* del año 2011 (NIL-UCM.1) y otra usando la base de datos de *SNOMED-CT* del año 2012 (NIL-UCM.2).

Los resultados obtenidos se presentan en los siguientes cuadros, y la presentación sigue la manera siguiente, en primer lugar se presenta el mejor de la tarea, a continuación nuestros resultados y finalmente el que quedo en peor posición

Tarea 1A

En esta tarea, se evaluaba la detección de los conceptos en el texto, es decir, únicamente debíamos indicar qué elementos aparecían y cuáles que no, sin llegar a realizar una clasificación.

Sin anotaciones externas, de manera estricta			
Equipo, País	*Precision*	*Recall*	*F1-Score*
UTHealth_CCB.2, UT, USA	0.800	0.076	0.750
NIL-UCM.2, Spain	0.617	0.4626	0.504
NIL-UCM.1, Spain	0.621	0.4616	0.498
FAYOLA.1, VW, USA	0.024	0.446	0.046

Tabla 4.2: Resultados *CLEF*. Modo estricto. Tarea 1A

Modo estricto Para estos resultados, quedamos en posición 18 y 20 de un total de 27 participantes.

Sin anotaciones externas, de manera estricta			
Equipo, País	*Precision*	*Recall*	*F1-Score*
UTHealth_CCB.2, UT, USA	0.925	0.827	0.873
NIL-UCM.2, Spain	0.809	0.558	0.660
NIL-UCM.1, Spain	0.812	0.543	0.651
FAYOLA.1, VW, USA	0.504	0.043	0.079

Tabla 4.3: Resultados *CLEF*, modo relajado, tarea 1B

Modo relajado Para estos resultados, quedamos en posición 21 y 22 de un total de 28 participantes.

Tarea 1B

En esta tarea, partiendo de los resultados de la tarea anterior, se debían clasificar los resultados obtenidos, es decir, determinar el *CUI* de aquello que hemos detectado. Únicamente se miraba la *Accuracy*, y se obtuvieron los siguientes resultados:

Sin anotaciones externas, de manera estricta		
Equipo, País	*Accuracy (sn2012)*	*Accuracy (sn2011)*
NCBI.2, MD, USA	0.589	0.584
NIL-UCM.2, Spain	0.362	0.362
NIL-UCM.1, Spain	0.362	0.362
NCBI.2, MD, USA	0.006	0.006

Tabla 4.4: Resultados *CLEF*, modo estricto, tarea 1B

Modo estricto Para estos resultados, quedamos en posición 10 y 11 de un total de 19 participantes.

Sin anotaciones externas, de manera relajada		
Equipo, País	*Accuracy (sn2012)*	*Accuracy (sn2011)*
AEHRC.1, QLD, Australia	0.939	0.939
NIL-UCM.1, Spain	0.871	0.870
NIL-UCM.2, Spain	0.850	0.850
UTHealth_CCB.1, UT, USA	0.728	0.772

Tabla 4.5: Resultados CLEF, modo relajado, tarea 1B

Modo relajado Para estos resultados, quedamos en posición 4 y 8 de un total de 19 participantes.

4.3.3. Otras tareas

4.3.3.1. Tarea 2

Esta segunda tarea, consistiría tendría como objetivo llevar a cabo la desambiguación de ciertos acrónimos presentes en los documentos de la tarea 1.

Para esta tarea, se nos facilitaba la ubicación de ciertos acrónimos y deberíamos decir cuál es era expansión teniendo en cuenta el contexto en el que apareciese el acrónimo.

Desafortunadamente, dadas las prisas por el poco tiempo disponible, se comprendió mal el objetivo de la tarea, pensando así que esta consistía en la detección de los acrónimos y en la determinación de la expansión correspondiente para cada uno de ellos. Esto tuvo como consecuencia que nos resultase imposible realizar correctamente y a tiempo la tarea y por tanto entregar la implementación y participar en ella.

4.3.3.2. Tarea 3

La especificación de tarea 3 indicaba que, dada una consulta, teníamos la responsabilidad de realizar una búsqueda y devolver aquellos informes que guardasen un mayor parecido con la consulta realizada.

Desde el *CLEF* se facilitó un conjunto de páginas web de las cuales nosotros pretendíamos extraer la información para poder realizar las búsquedas con mayor precisión.

Para la realización esta tarea, se modificó de nuevo lo implementado para la tarea 1, así como parte del trabajo que teníamos hasta el momento.

- Se dividieron los diferentes archivos facilitados para las búsquedas. Los originales contenían gran cantidad de páginas web, que debían ser separadas para poder realizar las búsquedas correctamente, por ello desarrollamos un programa que se encargase de separarlos, de eliminar también las partes de las páginas web que no resultarían útiles, y de añadir como primera línea de cada archivo la dirección de la página web correspondiente.
- Se cambió la parte de la tarea 1 relacionada con el almacenamiento de resultados de manera que existiese un archivo independiente por cada página leída, en lugar de uno que englobase todos.
- Se modificó el sistema para que, leyendo los documentos generados previamente, crease un índice sirviéndose de la librería *Lucene*.

  Dicho índice está compuesto como sigue:

  - `ID` de *SNOMED-CT* del concepto extraído
  - `Estado` del ID (`AFIRMADO`, `NEGADO` o `ESPECULADO`)
  - *`URL`* del archivo

  Los dos primeros darían la posibilidad de realizar las consultas creando el índice, y el *URL* estaría asociado al resultado obtenido.

- De nuevo con la ayuda de *Lucene* se implementó un método que, utilizando un *parser* para leer la consulta, una función de similitud y el índice mencionado en el punto anterior, realizase una búsqueda permitiendo así recuperar los documentos deseados.

Dificultades encontradas

Durante el desarrollo de esta última tarea nos encontramos con ciertos obstáculos, debido a la gran cantidad de información a procesar. Estamos hablando de unos 9 GB de información en texto plano que suponían cerca de 500 000 documentos distintos.

Si consideramos que el tiempo de procesamiento de cada uno de ellos es de 3 o 4 minutos aproximadamente, esto implicaría que para generar la información de la que se nutriría el índice de *Lucene* se tardarían, al menos, alrededor de

$$\frac{\left(500\,000\ documentos * 3\ \frac{minutos}{documento}\right)}{\left(1440\ \frac{minutos}{día} * 4\ ordenadores\right)} \backsimeq 260\ días,$$

siempre y cuando el trabajo fuese repartido entre los 4 ordenadores con que podíamos contar (uno por cada miembro del equipo de trabajo y uno extra por el director de proyecto). Esto conllevaría una cantidad inabarcable de tiempo dadas las circunstancias, especialmente si prestamos atención al hecho de que disponíamos de un plazo de 7 días para la realización de la tarea.

Capítulo 5

Arquitectura del sistema

5.1. Introducción

En este capítulo procederemos a explicar brevemente cuál es la arquitectura del sistema, dando al lector una visión general de la misma, y describiendo de manera más detallada los componentes principales. Está dirigido principalmente a aquellos desarrolladores que puedan estar interesados en conocer la estructura interna del proyecto ya sea para saber más sobre ella porque pretendan ampliar su funcionalidad o porque tengan intención de implementar una aplicación que utilice nuestro sistema, centrándonos principalmente en esto último.

Hemos tratado de omitir aquellos detalles no demasiado relevantes, detallando cómo a partir de la funcionalidad implementada, se ha construido un buscador de informes médicos y una interfaz que permita utilizarlo.

Las principales decisiones de diseño en cuanto a tecnologías utilizadas, no se incluyen en este capítulo, puesto que a lo largo de los capítulos Estado de la cuestión y Sistema de procesamiento de informes se explicó cuáles fueron las opciones consideradas a la hora de escoger las herramientas y recursos a utilizar, qué decisiones se tomaron y por qué, y cómo se aprovecharon las capacidades de cada una de ellas.

Por claridad, se ha dividido este capítulo en las siguientes secciones, además de la presente introducción:

- **Objetivos de la arquitectura y restricciones.** Por un lado declara escuetamente cuáles son los objetivos que tenemos en mente a la hora de diseñar la arquitectura, y por otro supone un recordatorio de cuáles han sido las tecnologías y librerías utilizadas y por tanto cuáles son las restricciones de uso del sistema.
- **Vista lógica.** Es la sección principal del capítulo, a lo largo de ella se presentarán y explicarán diversos diagramas de clases desarrollados con la herramienta de modelado *IBM Rational Software Architect Standard Edition 7.5*, de modo que se pueda entender cuál es la responsabilidad de cada uno de los elementos que constituyen la arquitectura y sus relaciones con el resto de componentes del sistema. Cabe destacar que, por claridad, no serán mostradas todas las relaciones existentes, sino únicamente aquellas que hemos considerado más importantes.

5.2. Objetivos de la arquitectura y restricciones

5.2.1. Objetivos

Es posible que una vez desarrollado un sistema aparezcan en un futuro nuevos desarrolladores que pretendan retomar su implementación, ampliando su funcionalidad o desarrollando otros sistemas tomando el primero como base.

De la misma manera que nosotros hemos partido de un software previo, cabría esperar que otros pudiesen hacer lo mismo con el proyecto que hemos llevado a cabo. Además cualquier sistema es susceptible a cambios, por lo que en la medida de lo posible un determinado software debe estar preparado para que se le introduzcan modificaciones con relativa sencillez. Así, debimos mantener esto en mente a la hora de especificar la arquitectura de sistema, siempre teniendo en cuenta que nos resultó imposible diseñarla desde cero, por tomar como base la del proyecto "Procesador Automático de Informes Médicos".

Por otro lado, tuvimos que considerar cuáles eran las mejores opciones a escoger en cuanto a herramientas a utilizar, importando, en nuestro caso, más la efectividad que la eficiencia. Esto se debe a que toma más peso el hecho de que los resultados sean aceptablemente buenos, que el tiempo que se tarda en obtenerlos.

5.2.2. Restricciones

El sistema, requiere unas condiciones mínimas para su correcta utilización. Si bien cuando se distribuye el software del sistema, se distribuye con todas las librerías actualizadas, creemos necesario que se sepa que librerías y en que versión se encuentran en el sistema.

- El sistema requiere un sistema operativo de 32 o 64 bits. Se admiten, tanto las ejecuciones en sistemas *Windows* como *GNU-Linux*.
- Se requiere el *Java Runtime Environment* o *JRE* de *32 bits*, versión 1.6 o superior. Si el sistema es *GNU-Linux*, también se admite el *JRE* de 64 bits.
- Es obligatorio que durante la ejecución del sistema, estén funcionando los tres servidores de MetaMap: *skrmedpostctl, wsdserverctl* y el *mmserver12* con el idioma correcto.
- Las librerías utilizadas junto con su versión son las siguientes:
  - *Hunspell*[1]: En su versión 1.3.2
  - *OpenNLP*[2]: En su versión 1.5.3
  - *Lucene*[3]: En su versión 4.0.0
  - *SQlite*[4]: En su versión 3.7.17

[1] http://hunspell.sourceforge.net/
[2] http://opennlp.apache.org/
[3] http://lucene.apache.org/core/mirrors-core-latest-redir.html?
[4] http://www.sqlite.org/

- *JDom*[5]: En su versión 2.0.5
- *MetaMap Java API*[6]: En la versión del año 2012.
- *PhoneticCodes*[7]: En su versión 1.7

5.3. Vista lógica

5.3.1. Visión general

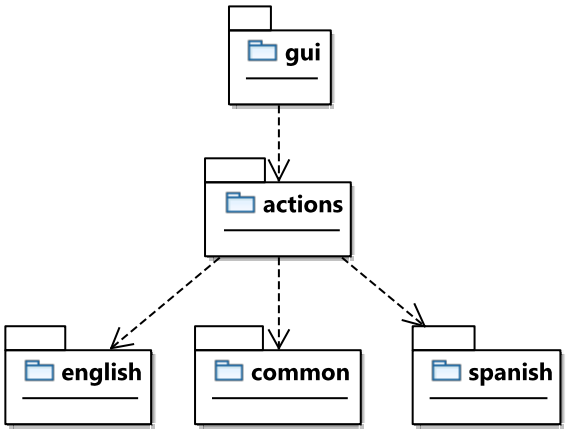


Figura 5.1: Paquetes principales del sistema

Recordemos que el sistema implementado permite procesar y extraer la información de informes médicos redactados tanto en inglés como en español. Teniendo esto en cuenta hemos dividido la funcionalidad del sistema en cuatro paquetes:

- El paquete `english` contiene todas aquellas clases que se encargan del procesamiento de informes exclusivamente en inglés. Es simétrico al paquete `spanish`.
- El paquete `spanish` recoge todas las clases y responsables del tratamiento de informes únicamente en español. Es simétrico al paquete `english`.
- El paquete `common` contiene la parte de la funcionalidad que es común a ambas partes del sistema, y que por tanto serán utilizadas para extraer información tanto de informes escritos en inglés como de informes escritos en español.
- El paquete `gui` engloba todas las clases encargadas de mostrar una interfaz al usuario que le permita ejecutar la aplicación y hacer uso de su funcionalidad de manera sencilla.

[5] http://www.jdom.org/
[6] http://metamap.nlm.nih.gov/#MetaMapJavaApi
[7] http://commons.apache.org/proper/commons-codec/

- El paquete `actions` contiene aquellas clases que permiten servir de fachada para la interfaz. De esta forma, y siguiendo el patrón *Facade*, el paquete `gui` puede utilizar las clases del paquete `actions` para abstraerse de la funcionalidad proporcionada por las clases encargadas del procesamiento y búsqueda de informes (que se encuentran en los paquetes `english`, `spanish` y `common`) y acceder de manera sencilla a ella sin necesidad de conocer qué hay por debajo. Esto permite que cualquier modificación que se realice a nivel de procesamiento no afecte a la interfaz directamente, sino que los cambios deberán verse reflejados en las clases del paquete `actions`. Esta situación es la que podemos ver ilustrada en la figura .

5.3.2. Paquete `gui`

Como hemos explicado previamente, en este paquete se encuentran las clases específicas de la interfaz gráfica del sistema. Esperando que sirva de ayuda para la comprensión de la arquitectura, cabe destacar que estas clases siguen un convenio que determina que, en caso de que una de ellas extienda la clase `JPanel` entonces terminará con el sufijo `_p`, mientras que, en caso de que extienda la clase `JFrame` el nombre de la clase terminará con el sufijo `_f`.

A continuación procedemos a detallar el cometido de cada una de las clases principales de este paquete, cuyas relaciones fundamentales incluimos en la figura 5.2.

- `MainFrame_f`*:* La interfaz principal contiene un único formulario cuyo panel se irá modificando en función de en qué situación nos encontremos y cuáles hayan sido los pasos ejecutados hasta el momento. Será, por tanto, la clase encargada de mostrar la pantalla principal del sistema al comienzo de la ejecución y de comprobar si se ha dado un valor inicial al directorio de trabajo de la aplicación. Dado que la existencia de un directorio de trabajo es imprescindible, en caso de que dicha acción no se haya realizado correctamente, esta clase procederá a finalizar la ejecución.

- `MainPanel_p`: Supone el panel principal de la aplicación, y que da acceso a diferentes pantallas del sistema. Estas pantallas son las que vienen representadas por las clases `OpenAndProcess_p`, `Opened_p`, `Searcher_p`, `Found_p` y por último `Settings_p`.

- `OpenAndProcces_p`: Esta clase permite mostrar la pantalla que da la posibilidad abrir informes ya procesados, procesar otros nuevos o ver la tabla que contiene el último conjunto de informes cargados por la aplicación. La forma en que realiza estas acciones la veremos detallada más adelante.

- `Opened_p`: La clase `OpenedProcess_p` es la que se encarga de presentar al usuario el último conjunto de informes que han sido cargados por la aplicación mediante alguna de las posibles acciones que pueden ser ejecutadas desde el panel `OpenAndProcess_p`. Desde ella podremos seleccionar uno de dichos informes y visualizarlo como se explicará en el apartado .

- `Searcher_p`: La misión de esta clase es la de permitir al usuario realizar búsquedas a través de una interfaz que le de la posibilidad de usar cualquiera de los modos del buscador: búsqueda mediante formulario y búsqueda de informes similares a uno dado. Una vez recuperados los informes, estos se nos mostrarán a través del `JPanel Found_p`.

- `Found_p`: Gracias a esta clase podremos visualizar el último conjunto de informes recuperados por el buscador. Desde la vista que nos presenta podremos visualizar cada uno de ellos utilizando las clases que detallaremos en párrafos posteriores.
- `Settings_p`: La última de las clases que nos tomamos la licencia de denominar principales del paquete `gui`, es esta, la encargada de mostrar al usuario aquellos elementos visuales que le permitirán ajustar ciertos parámetros de configuración de manera más sencilla y cómoda que como se explica a lo largo del apéndice B.

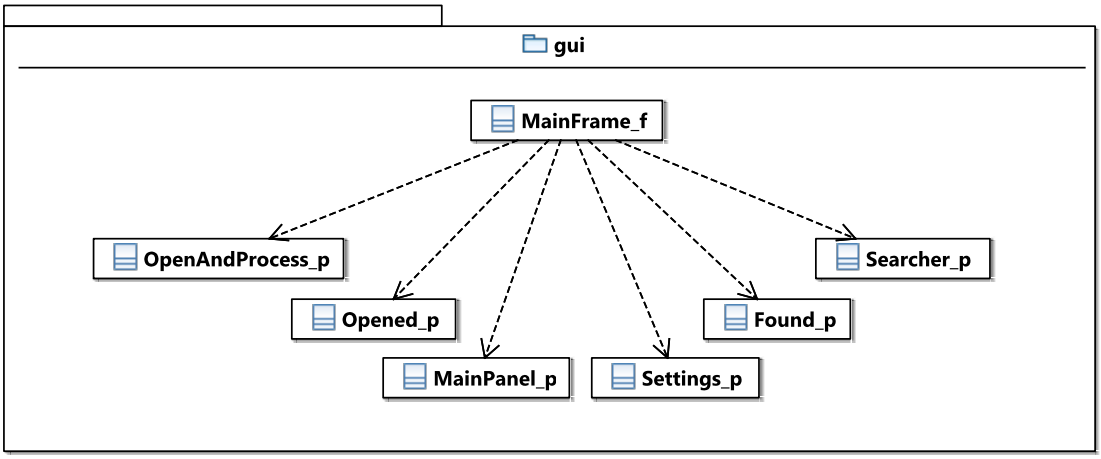


Figura 5.2: Clases principales del paquete `gui`

5.3.3. La clase `ReportsTable_p` y sus relaciones

En el apartado 5.3.2 se introdujo el hecho de que las clases `Opened_p` y `Found_p` permitirían mostrar una pantalla con diversos conjuntos de informes; en el caso de `Opened_p` dicho conjunto estaría formado por los últimos que fueron cargados por el sistema y en el caso de `Found_p` el conjunto lo compondrían aquellos informes que hubiesen sido devueltos como resultado de una búsqueda.

A la hora de visualizar dichos informes, la forma de presentarlos por pantalla dependerá de en qué situación nos encontremos.

En el caso de `Opened_p` si la última acción realizada con éxito desde `OpenAndProcess_p` consistió en:

- importar uno o varios informes desde uno o varios archivos con formato `.txt` o `.xml`, o procesar un informe desde un archivo `.mxml`, entonces:

  el informe seleccionado se nos mostrará en una nueva ventana que contará con una doble vista, mediante un `JSplitPane` de tipo `ReportsSplitPane_p`. Mientras que en uno de los lados podremos visualizar el archivo `.mxml` obtenido tras el mapeo o la transformación del archivo original, y que contará por tanto con las secciones detalladas en el apartado 3.2.1; en el otro lado podremos ver el contenido del informe procesado. Estas dos vistas serán logradas gracias al uso de las clases `Report_p` y

`ReportWithConcepts_p`. La primera de ellas se encargará de mostrar el contenido de un informe a partir de un `ReportModel`, mientras que la segunda partirá de un `ConceptsReport`.

- abrir un informe ya procesado con formato `.pxml`, entonces:

  el informe seleccionado se nos presentará en una ventana nueva con un único `JPanel` de tipo `ReportWithConcepts_p`. Al no poder estar seguros de que se cuente con el archivo .mxml asociado, intacto, únicamente se mostrará el informe de esta manera construyendo el `JPanel` a partir de un `ConceptsReport`.

En el caso de `Found_p` siempre se utilizará la segunda vista, dado que como en dicho caso, nunca podremos estar seguros de que el archivo `.mxml` se conserva intacto.

Por último cabe señalar, que puesto que las diferencias existentes entre `Opened_p` y `Found_p` son mínimas, toda la funcionalidad común se encuentra recogida en la clase abstracta `ReportsTable_p`, de manera que, como detallamos en la figura 5.3, ambas la extiendan según lo precisen.

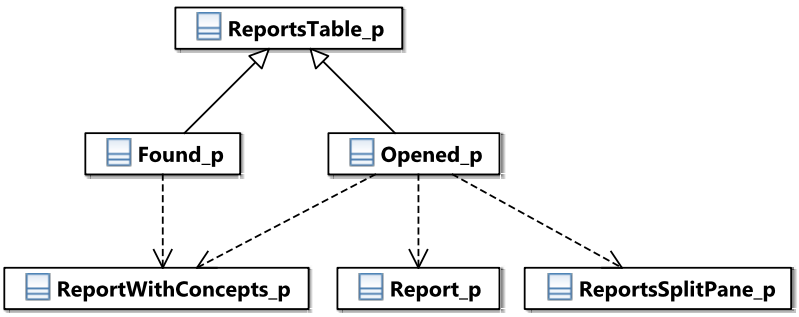


Figura 5.3: Clases de encargadas de gestionar la visión de informes

5.3.4. La clase `Settings_p` y sus relaciones

En el sistema implementado, se pueden realizar gran cantidad de ajustes en diversos archivos de configuración cuyo contenido y cometido se describen en el apéndice . Sin embargo, con la intención de que el uso del software desarrollado resulte más amigable, se han incluido como parte de él una serie de clases que permiten modificar a través de la interfaz algunos de ellos.

Dichos ajustes se centran en dos características de gran importancia:

se pueden realizar un conjunto de ajustes para una correcta y mejor detección de los conceptos. Los ajustes permitidos son los siguientes:

- El idioma, tanto de la propia interfaz como el del procesador a utilizar (i. e. en el que se encuentran redactados los informes a procesar).

- El mapeo y la transformación de informes médicos a partir de archivos `.txt` y `.xml`.

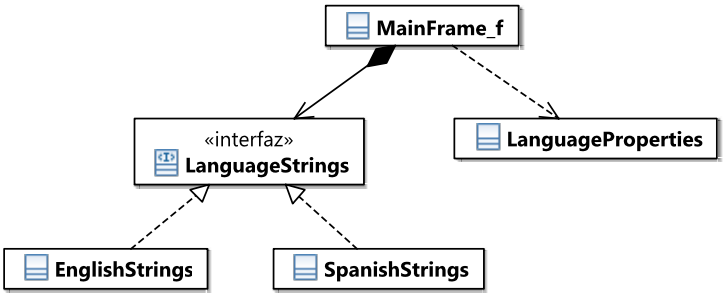


Figura 5.4: Relaciones entre las clases que permiten configurar y usar el idioma

Comenzando por el primero de los puntos, entraremos a explicar en primer lugar cómo se determina el idioma de la interfaz y cómo se da valor a todas las cadenas de texto que se muestran a través de ella. Para ello seguiremos el diagrama adjunto en la figura 5.4. La protagonista en este caso es `LanguageStrings`. Esta interfaz detalla una serie de métodos. Cada uno de ellos devuelve un objeto tipo `String` que representa una de las cadenas de texto a mostrar por la interfaz, y según un convenio propio, se deberá encontrar declarado como `public String get<NombreQueIdentifiqueLaCadenaDeTexto>_str()`. A raíz de esta interfaz, surgen dos clases `EnglishStrings` y `SpanishStrings` que la implementan debidamente para el idioma correspondiente. Este mecanismo permite que la traducción de la interfaz a otro idioma sea una tarea realmente sencilla, consistiendo únicamente en la creación de una nueva clase para el idioma que interese de modo que implemente la interfaz `LanguageStrings`.

Por otro lado contamos con la clase `LanguageProperties`, que permitirá leer y modificar el archivo de configuración `language.properties`, y por tanto permitirá acceder tanto a la propiedad que fija el idioma de la interfaz como a la que fija el idioma del procesador a utilizar, el que trata con informes en inglés o el que permite extraer información de informes en español.

Tal como vemos en el diagrama de la figura 5.4, y a pesar de que nos encontramos en un apartado dedicado a la clase `Settings_p`, es la clase `MainFrame_f` quien tiene acceso al archivo de configuración `language.properties` a través de la clase `LanguageProperties`. Esto es así porque dado que es la clase principal, se consideró oportuno que fuese ella quien consultase la propiedad asociada al idioma de la interfaz con el fin de asignar un valor inicial de una u otra manera al objeto de tipo `LanguageStrings` que posee. Sin embargo a pesar de ello, es a partir de la clase `Settings_p` de quien podemos consultar o variar de manera visual el valor de dicha propiedad. Este `JPanel` delegará la acción asignada por el usuario al formulario principal `MainFrame_f`, de forma que así se modifique también en valor del objeto de tipo `LanguageStrings` que contiene.

Además del idioma de la interfaz, desde el menú de ajustes es posible modificar el idioma para el que se desea que se procesen los archivos, pero como en el caso contrario estas acciones de modificación serán delegadas al formulario principal.

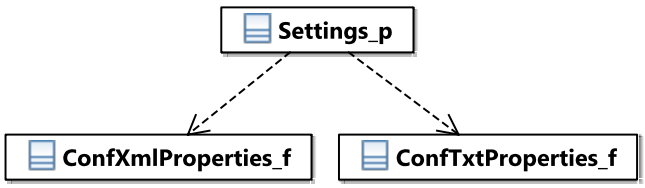


Figura 5.5: Clases relacionadas con los ajustes de la aplicación

Centrándonos ahora en el mapeo y transformación de informes, hacemos en primer lugar referencia al apartado para que el lector pueda conocer más sobre los archivos `xml.properties` y `txt.properties`. Una vez conocido su cometido, podemos decir que su contenido puede modificarse gracias a las clases `ConfXmlProperties_f` y `ConfTxtProperties_f` respectivamente. Para acceder a ambas, basta con acceder a la clase `Settings_p` que hace uso de ellas.

5.3.5. La clase `ProcessorActions` y sus relaciones

La interfaz y el procesador son clases y paquetes totalmente independientes, pero deben comunicarse de alguna manera, para ello, tal como hemos explicado al comienzo de la sección, contamos con el paquete `actions` cuyas clases sirven de fachada a la interfaz aislándola de manera que pueda llevar a cabo todas las acciones posibles de manera mucho más sencilla en comparación con como lo tendría que hacer, si accediese directamente a la funcionalidad de los paquetes `english`, `spanish` y `common`.

Tal como muestra el diagrama de la figura 5.6, gracias al uso de `ProcessorActions` podemos acceder a las siguientes clases:

- `Processor`, que representa, como es de suponer la clase que permite procesar informes. Para tratar un informe, el procesador nos solicitará una ruta de origen y una ruta de destino que identificarán por un lado al informe con formato `.mxml` que habrá de ser procesado y por otro al informe ya procesado con formato `.pxml`.

  Para poder proporcionar estas rutas, bastará con pedir los datos al usuario a través de la interfaz, y concretamente a través de `OpenAndProcess_p`, de forma que podamos obtener información acerca de cuál es el archivo a procesar. Teniendo en cuenta que el archivo que representa el informe procesado tendrá el mismo nombre que este, pero con extensión .pxml, ya habremos reunido lo necesario para llamar al método correspondiente de la clase `Processor`.

  Tal como podemos ver, la clase `Processor`, es una clase abstracta, que cuenta con las subclases `EnglishProcessor` y `SpanishProcessor` que permitirán procesar unos informes otros en función de idioma. Para poder conocer cuál es el idioma que el usuario desea utilizar, `ProcessorActions` hará uso de la clase `LanguageProperties` que nos da acceso al archivo de configuración `language.properties`.

  Por otro lado, para poder llevar a cabo este procesamiento, la clase `Processor` mantendrá las relaciones que vemos en la figura , con las clases `ReportModel` y `MedicalReport` de manera que tras procesar un informe se obtendrá un objeto de tipo `MedicalReport`. Cabe señalar que este tipo de objeto podrá ser obtenido mediante

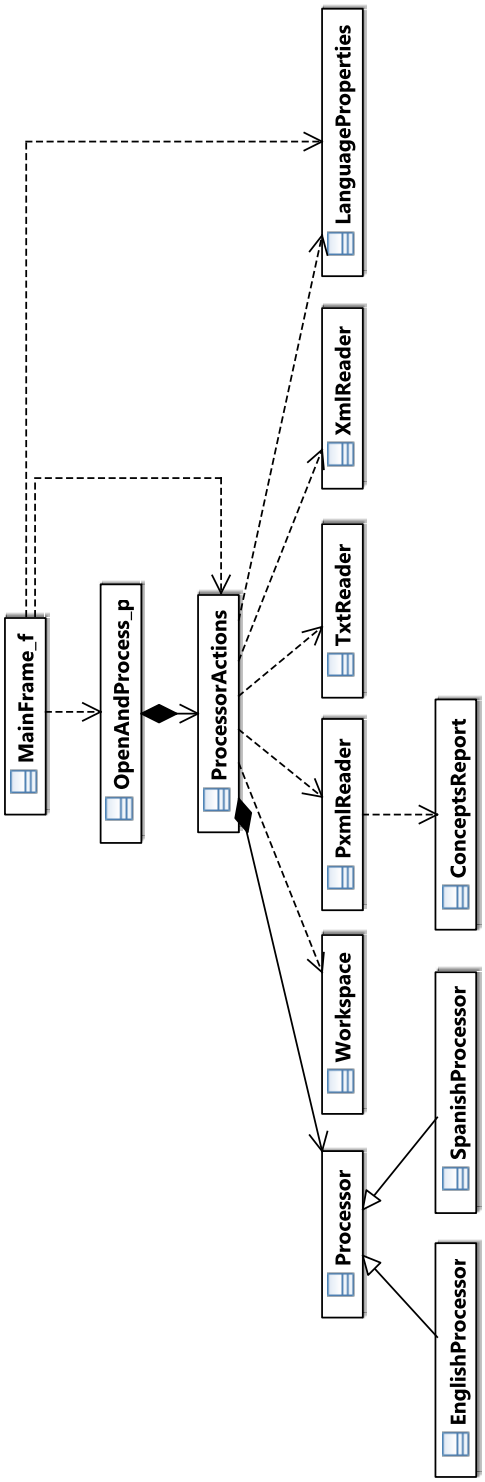


Figura 5.6: Relaciones de la clase `ProcessorActions`

diferentes métodos, no solo tras procesar, ya que supone la representación básica de un informe. Pero independientemente de esto, lo que nos interesa en este caso, es que a partir de un `MedicalReport` siempre podremos obtener un `ReportModel` que contenga diversos datos, como el contenido de las secciones del propio informe.

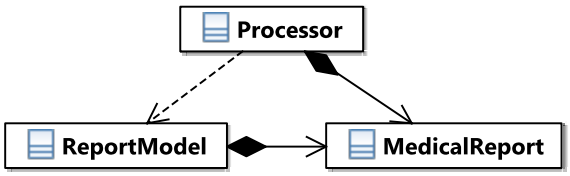


Figura 5.7: Representación informe médico

- `Workspace`. Esta clase representa el directorio de trabajo de la aplicación. En dicho directorio se almacenarán todos los archivos generados por ella, y además se crearán, en caso de que no existan, diferentes carpetas para cada tipo.

  Así, obtendremos una carpeta para los archivos que se obtengan tras la importación, y que por tanto serán de tipo `.mxml`, otra para los informes procesados, y que tendrán extensión `.pxml` y por último una carpeta que almacenará el índice generado a partir de un conjunto de informes `.pxml` y que servirá para poder, posteriormente, realizar búsquedas sobre ellos.

  La clase que representa este directorio de trabajo sigue el patrón *Singleton* y cuenta con diversos métodos de tipo `static` que, no solo dan la posibilidad de recuperar el valor del propio `Workspace`, sino también el de las rutas correspondientes a las carpetas que acabamos de describir.

  Deberá dársele un valor inicial antes de poder realizar cualquier otra acción, y ello se hará desde la clase `ProcessorActions` por orden de `MainFrame_f`. Esta última como ya hemos mencionado, comprobará que efectivamente el usuario ha seleccionado correctamente un directorio de trabajo, y en caso de que no sea así, procederá al cierre de la aplicación.

- `PxmlReader`, que dará la posibilidad de, como su propio nombre indica, leer archivos de tipo `.pxml`. Esta clase generará como resultado un objeto de tipo `ConceptsReport` que contendrá todos los datos almacenados en dicho informe procesado. Estas dos clases, serán fundamentales en la ejecución de la aplicación, ya que todo el sistema gira en torno a los informes con extensión `.pxml` y estas clases permitirán mostrar al usuario dichos archivos y realizar procesos de indexación de informes y búsqueda de los mismos.

- `TxtReader`. Esta clase cobrará importancia en el momento en que debamos importar uno o varios informes desde un archivo con formato `.txt` para que sean procesados. En ese momento, y gracias a esta clase y a su método `createMxml` podremos crear un informe que posea la estructura requerida para que pueda ser procesado.

- `XmlReader`. Esta clase tomará uno de los papeles protagonistas si el usuario que esté ejecutando la aplicación, decide importar uno o varios informes desde archivos `.xml`, con el fin de procesarlos. Cuando esto suceda, y mediante la invocación del método estático `createMxml`, de esta clase, lograremos crear un archivo `.mxml` que represente el informe a procesar.

- `LanguageProperties`. La relación que mantienen estas dos clases ya ha sido comentada en un punto anterior. La finalidad es que `ProcessorActions` pueda conocer el idioma que figura en el archivo de configuración correspondiente, `language.properties` para que pueda crear correctamente el procesador que le interese usar (`EnglishProcessor` o `SpanishProcessor`) para tratar los informes y extraer de ellos su información.

  En ese mismo diagrama de la figura 5.6, podemos apreciar la relación que hay entre esta clase y la clase `MainFrame_f`. Esta relación existe con el objetivo de que todas las cadenas de texto que son mostradas a través de la interfaz puedan mostrarse en el idioma apropiado. Además, a la hora de modificar el idioma desde el menú de ajustes el panel `Settings_p` delegará la responsabilidad en `MainFrame_f`.

5.3.6. La clase `ReportProcessor` y sus relaciones

El procesado de los informes es solicitado por parte de la interfaz gráfica a la clase `ProcessorActions` y desde esta se ordena ejecutar la acción correspondiente a los paquetes `english`, `spanish` y `common`. Además, como anticipábamos anteriormente, existen dos procesadores independientes, uno para los informes médicos que están redactados en español y otro para los que han sido escritos en inglés. Así pues este procesador se encargará de llamar a su vez al método `processReport()` de la clase `ReportProcessor` del paquete correspondiente, y una vez obtenido el resultado proporcionado por este, en forma de `MedicalReport`, generará el archivo `.pxml`.

Esta simetría entre paquetes y clases podemos apreciarla en las figuras 5.8 y 5.9.

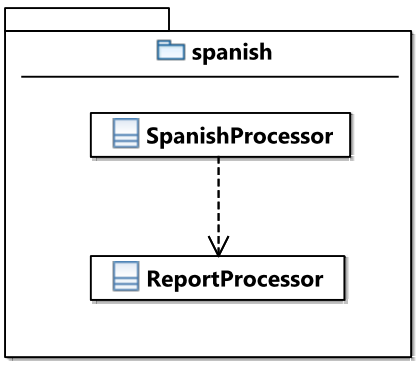


Figura 5.8: `ReportProcessor`. Paquete `spanish`

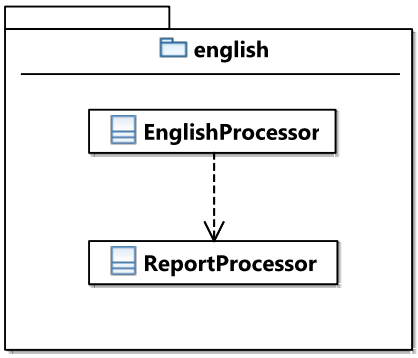


Figura 5.9: `ReportProcessor`. Paquete `english`

A continuación entraremos más en detalle sobre las relaciones que mantienen las clases `ReportProcessor` de ambos paquetes con el resto de clases de los mismos, y sobre las tareas que realiza. Por simplicidad, y dada la simetría que hemos comentado, mostraremos únicamente el diagrama correspondiente a uno de ellos, ya que el otro resultaría equivalente. El diagrama del que hablamos es el adjunto en la figura 5.10.

Esta clase es la que se encarga de todo el procesamiento en sí mismo, realizando las tareas detalladas a lo largo de la sección 3.3 y que por tanto no entraremos a describir de nuevo. Es por ello, que se comunica con diversos paquetes, estando encargados cada uno de ellos de la ejecución de una de las fases en que se descompone el procesamiento de un informe. Dicho esto pasaremos pues a identificar cada una de dichas fases con la clase correspondiente con que está relacionada el `ReportProcessor`., recordando cuáles eran los apartados en los que explicábamos en profundidad el funcionamiento de cada una de ellas tanto en inglés como en español.

- `Spellchecker`. Implementa el comportamiento descrito en las subsecciones 3.3.1.1 y 3.3.2.1.
- `AcronymsExpander`. Es la clase responsable de la fase de expansión de acrónimos cuyo funcionamiento está contenido en las subsecciones 3.3.1.2 y 3.3.2.2.
- `NegExMetaMap/NegEx`. Es la clase encargada de detectar aquellas frases que se encuentran negadas con el fin de poder determinar el estado de los conceptos contenidos en ella. Más sobre esta fase en los apartados 3.3.1.3 y 3.3.2.3.
- `Conditional`. Es la clase encargada de detectar aquellas frases que se encuentran negadas con el fin de poder determinar el estado de los conceptos contenidos en ella. Si desea conocer más sobre el funcionamiento de esta fase puede dirigirse a las subsecciones 3.3.1.4 y 3.3.2.4.
- `ConceptsPhaseMetaMap`. Es la clase que implementa la detección de conceptos contenidos en las bases de datos creadas a partir de *UMLS*. Los detalles sobre el funcionamiento de esta fase están en los apartados 3.3.1.5y 3.3.2.5.

Adicionalmente podemos observar la relación que mantiene con la clase `MedicalReport`, que viene dado por el hecho de que el propio objeto de este tipo, representará el informe que está siendo procesado.

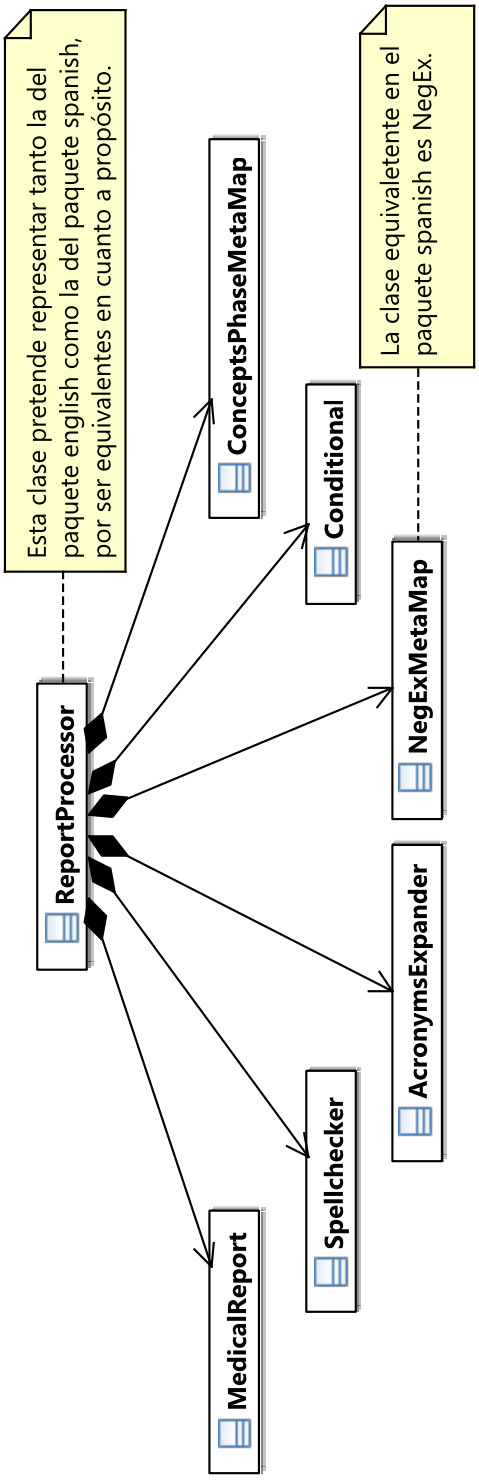


Figura 5.10: Relaciones de `ReportProcessor`

5.3.7. La clase `SearcherActions` y sus relaciones

Desde la interfaz gráfica el usuario que esté utilizando la aplicación puede solicitar, a la hora de abrirlos, la creación de un índice para un conjunto de informes `.pxml`. Por otro lado como aplicación que permite aprovechar la representación obtenida mediante el extractor de información, se ha desarrollado, como ya deberá conocer el lector, un buscador.

Sin embargo según nuestro juicio, hay ciertos aspectos que no tendría por qué conocer la interfaz. Por este motivo procedemos a hablar a continuación de la clase `SearcherActions`, contenida en el paquete `actions`, y que al igual que `ProcessorActions` sirve como fachada de los paquetes `english`, `spanish` y `common` con el fin de que la interfaz pueda abstraerse del funcionamiento interno de estos.

Así, tanto la clase `OpenAndProcess_p`, a través de cuyos componentes podremos mandar crear el índice, como la clase `Searcher_p`, que será la responsable de la presentación por pantalla del menú de búsqueda, podrán, mediante un objeto de tipo `SearcherActions`, realizar fácilmente todas aquellas acciones que necesiten relacionadas con la búsqueda de informes.

La creación del índice y la realización de consultas requerirán la presencia de objectos de tipo `IndexLuceneCreator`, `Workspace` y `PxmlReader`.

- `IndexLuceneCreator`, haciendo uso de *Lucene*, será el responsable de realizar todo el trabajo, i. e. la construcción del índice y la realización de consultas sobre el mismo.

- `Workspace`. Esta clase vuelve a aparecer, ya que a la hora de crear el índice, y realizar consultas será necesario indicarle el destino del mismo a `IndexLuceneCreator`, por lo que `SearcherActions` requerirá la colaboración del `Workspace`.

- Por último, a la hora de realizar las consultas, necesitaremos almacenar los resultados encontrados, y esto se hará guardando objetos de tipo `ConceptsReport` para cuya obtención necesitaremos hacer uso de `PxmlReader`.

Podemos observar también en el diagrama la relación existente entre `IndexLuceneCreator` y `PxmlReader`. Esta relación tiene como causa, la necesidad evidente que posee la primera de poder leer informes en formato `.pxml` para la creación del índice, y para la obtención de informes similares a uno dado.

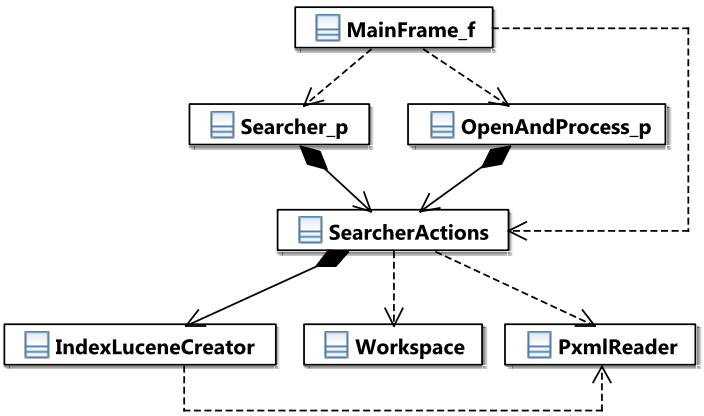


Figura 5.11: Relaciones de la clase `ProcessorActions`

5.3.8. Enmascarado de MetaMap

Debido a que se utiliza MetaMap desde diversos puntos de la aplicación, vimos oportuno ocultar su funcionamiento interno para que todos consiguiesen el resultado adecuado sin necesidad de preocuparse por las acciones que se deben llevar a cabo para obtenerlo.

De este modo se desarrollaron dos clases dependientes del idioma siguiendo el patrón *Singleton*. Así, no solo permitirían servir de conexión entre MetaMap y la aplicación, sino que además podríamos estar seguros de que, estando asociada cada una a una conexión con MetaMap, no podría existir nunca más de una conexión simultánea por idioma.

Una vez logrado implementadas las clases conseguimos la estructura que se puede visualizar en la figura , de manera que harían uso de ellas las siguientes clases (nótese que algunas de ellas son las que aparecían en la figura 5.10 del apartado La clase `ReportProcessor` y sus relaciones):

- `AcronymsExpanderMetaMap`, cuyas funciones serán solicitadas desde la clase `AcronymsExpander` con el objetivo, obvio de expandir los acrónimos.
- `NegExMetaMap`, cuyo cometido será encargarse de la detección de frases negadas.
- `ConceptsPhaseMetaMap`, que tiene como propósito, como cabe esperar, detectar los conceptos presentes en las bases de datos que estemos utilizando con MetaMap.
- `SercherActions`. A pesar de que vemos que esta clase no formaba parte del diagrama de la figura Relaciones de `ReportProcessor` por no participar en las diferentes fases que componen el procesamiento de un informe, será necesario que haga uso de `MetaMapSingleton` por ser el encargado de obtener los conceptos médicos presentes en los diferentes campos del formulario de búsqueda. Gracias a ello, podremos construir una consulta que se realizará gracias a las clases presentadas en el apartado 5.3.7.

A su vez, para poder configurar el lanzamiento de MetaMap correctamente contamos con la clase `MetaMapProperties` que otorga la posibilidad de leer y escribir sobre el archivo de configuración `metamap.properties` (detallado en el apéndice Configuración).

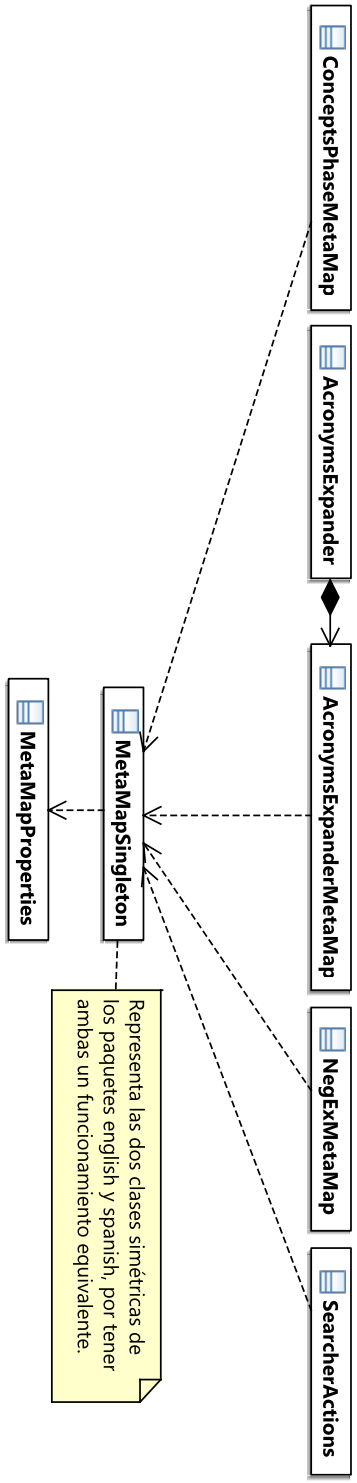


Figura 5.12: Comunicación con MetaMap

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Tras un curso académico desarrollando este proyecto podemos decir que hemos cumplido los objetivos planteados implementando un sistema que extrae información de un informe médico utilizando la herramienta MetaMap. La información extraída es clasificada en función de como aparece, si afirmada, negada o especulada y también atendiendo al apartado del informe en que aparece. A su vez, utilizando esta información, y este sistema hemos desarrollado un buscador para los informes analizados, que pueda ilustrar las aplicaciones y usos que se le podrían dar.

Así mismo podemos decir que hemos modificado nuestro sistema con el fin participar en el internacionalmente conocido *CLEF*, concretamente en la edición del presente año 2013.

6.1.1. Evaluación

Como método de evaluación, hemos determinado utilizar los resultados del *CLEF* del año 2013.

En estas conferencias se nos evaluó y se nos comparó con otros sistemas desarrollados por el resto de equipos participantes. Los resultados finales fueron los siguientes:

- Para la tarea 1A de manera estricta, se obtuvo como mejor *precision*: 0.617, como mejor *recall*: 0.4626 y como F1-score un total de 0.504, situándonos en posición 18 de 27 participantes.
- Para la tarea 1A de manera relajada, se obtuvo como mejor *precision*: 0.809, como mejor *recall*: 0.558 y como F1-score un total de 0.660, situándonos en posición 21 de 28 participantes.
- Para la tarea 1B de manera estricta, se obtuvo como mejor *Accuracy* (sn2012): 0.362 y como *Accuracy* (sn2011): 0.362, situándonos en posición 10 de 19 participantes.
- Para la tarea 1B de manera relajada, se obtuvo como mejor *Accuracy* (sn2012): 0.871 y como *Accuracy* (sn2011): 0.870, situándonos en posición 4 de 19 participantes.

6.2. Ampliaciones potenciales y trabajo futuro

Nuestro sistema, puede ser utilizado como base de otro tipo de aplicaciones con un estilo parecido, aunque también se puede ampliar con unas mejoras que creemos que aumentarían la precisión del sistema.

6.2.1. Protección de datos

Como hemos comentado en otras ocasiones, en este proyecto se trabaja con información muy sensible cuyo tratamiento puede resultar muy complejo. Así, los informes de los que disponemos, no contienen información que permitan identificar a los pacientes y cuenta con ciertos campos protegidos según recoge la ley de protección de datos.

Si esta aplicación se utilizase en un hospital, entendemos que este problema de protección de datos quedaría resuelto, pues el hospital podría manejar los datos de los pacientes libremente, siempre de manera confidencial. Dado que por el momento este no es el caso, convendría implementar una fase previa al análisis que se encargase de eliminar y corregir aquellos datos que estén clasificados como personales. De este modo la aplicación podría ser utilizada con conjuntos de informes que contuviesen originalmente información personal, pero que la propia aplicación se encargase de proteger o eliminar.

6.2.2. Detección de las especulaciones

Siguiendo algunas directivas de trabajo futuro indicadas en la memoria del año pasado, y tras leer ciertos artículos que trataban el tema, se creyó necesario incluir la fase de detección de las especulaciones. Si bien así se hizo, pensamos que probablemente esta se podría mejorar mejorar, especialmente si se contase con un gran conjunto de informes médicos sobre los que ejecutar un proceso de un aprendizaje automático.

6.2.3. Mejora de las consultas

Las consultas, actualmente, no pueden elaborarse de manera que resulten significativamente complejas, por lo que no se aprovecha todo el potencial que posee y nos ofrece *Lucene*. En caso de que se desease realmente tener un sistema que realizase búsquedas de relativa complejidad, el buscador desarrollado podría ampliarse para conseguirlo, ya que en nuestro caso, se trata tan solo de una aplicación, de entre todas las posibles que se podrían realizar, que permite probar la utilidad del sistema principal.

6.2.4. Utilización del código CIE

Cada informe, esta anotado manualmente por un médico con un código de Clasificación Internacional de Enfermedades o CIE, que permite conocer el asunto principal del informe por describir información relativa al estado de salud del paciente.

En las búsquedas y la extracción de los conceptos, la lectura e interpretación del CIE sería muy útil, puesto que podría utilizarse para mejorar la fase de desambiguación. Esto

sería así, debido a que este código permitiría conocer con qué rama de la medicina o con qué campo o ámbito médico se encuentran relacionados los conceptos que contiene el informe.

6.2.5. Extracción de estadísticas

Una posibilidad realmente interesante, comentada en alguna de las reuniones del director de proyecto con profesionales del hospital Clínico de Madrid, consistiría en realizar documentos estadísticos a partir de los informes obtenidos tras una búsqueda. Desarrollar esta aplicación habría resultado una tarea relativamente sencilla, pero lamentablemente debido al tiempo con que contamos no pudimos llevar esto a cabo. El objetivo, como comentamos, sería la obtención de resultados estadísticos sobre un conjunto de informes, como puede ser, porcentaje de enfermedades, porcentaje de síntomas, relación enfermedad-sexo o enfermedad-edad, así como la realización de las mismas estadísticas cuando se realice una búsqueda.

6.2.6. Ampliación a otros idiomas

La ampliación a cualquier otro idioma será mas o menos simple, únicamente hay que realizar las conversiones necesarias para cada una de las fases de las que se compone el proyecto:

Corrección de faltas ortográficas:

Para traducir esta parte del sistema bastaría con sustituir el diccionario utilizado por uno en el idioma concreto que interesase y con modificar el algoritmo fonético que se utiliza en la puntuación de sugerencias para que, de nuevo esté adaptado al idioma que deseemos.

Expansión de acrónimos

Para completar esta fase del procesamiento de informes en otro idioma, deberá completarse una base de datos con los acrónimos concretos que se deseen incluir para ese idioma, y ciertas reglas propias del idioma, que permitan llevar a cabo la desambiguación.

Detección de las negaciones

Si se desease traducir esta parte de la implementación, la persona encargada deberá modificar el algoritmo NegEx para que detecte las negaciones en el idioma correspondiente. Cabe la posibilidad de que únicamente sea necesario cambiar aquellas palabras clave asociadas a cada conjunto de negaciones que incluye el algoritmo y que son utilizadas por él.

Detección de las especulaciones

En caso de que se desease ampliar los idiomas para los que esta fase se encuentra disponible, el responsable de esta tarea, se vería obligado a modificar el algoritmo que hemos desarrollado para que detecte las especulaciones en el idioma elegido. De nuevo, cabe la posibilidad de que baste con cambiar aquellas palabras clave de cada conjunto de especulaciones utilizadas por el algoritmo. Cabe señalar que este método fue usado para obtener esta parte de la implementación en español, partiendo del inglés, y que se alcanzaron resultados satisfactorios.

Detección de los conceptos

En caso de que la persona encargada de realizar esta tarea no pudiese desarrollar las bases de datos que se utilizables con MetaMap[1][2], el cambio sería drástico, pues esta es la parte más compleja. En ese caso estaría obligado a desarrollar otro tipo de base de datos de conceptos que además debería incluir un sistema de desambiguación.

6.2.7. Otros proyectos

El sistema que hemos implementado podría ser aprovechado para desarrollar otro tipo de proyectos relacionados con el campo de la medicina y de los informes médicos, a continuación proponemos algunos ejemplos de sistemas que creemos que podrían resultar de utilidad.

Sistema de comprobación o de auto-anotación del código CIE

Este código, como ya se ha comentado, se utiliza en medicina para la clasificación de los informes clínicos, puesto que contiene la información más importante de ellos, por lo que desarrollar una aplicación que indique qué CIE debería asignarse a un documento dado, facilitará el trabajo al médico, pues no deberá buscar aquel que es más correcto en una base de datos o al menos podría realizarle sugerencias obteniendo así un ahorro de tiempo.

Buscador de pacientes para estudios clínicos

A pesar de que ya se es posible realizar una consulta o búsqueda que proporcione estos datos, cabría la opción de confeccionar un buscador que devolviese como resultado conjuntos de pacientes o personas que pudiesen resultar adecuados para la realización de ciertos estudios clínicos relacionados con medicamentos y enfermedades, asimismo se podría incluir un generador de resultados una vez concluido dicho estudio.

[1] En MetaMap, existen bases de datos de los siguientes idiomas: sueco, español, ruso, portugués, noruego, coreano, ingles, alemán y checo entre otros.

[2] No todos los idiomas disponen de una base de datos de SNOMED-CT actualizada y traducida, por lo que es posible que no sea factible el desarrollo de estas bases de datos.

Apéndice A

Manual de usuario

A.1. Requisitos previos

Para poder ejecutar el sistema que hemos desarrollado será necesario cumplir una serie de requisitos *software* previos. En concreto deberemos tener instalados los siguientes componentes:

- *Java Runtime Environment* o *JRE* de *32 bits*, versión 1.6 o superior[1].

  Para ello desde la propia página recomiendan, en lo relativo a requisitos *hardware*, contar un Pentium 2 a 266 MHz o un procesador más rápido con al menos 128 MB de RAM física. En caso de que se vaya a hacer uso del sistema operativo *Windows XP*, serán necesarios al menos 64MB de RAM en lugar de 128MB. Asimismo también será preciso disponer de un mínimo de 124 MB de espacio libre en disco.

  Enlace de descarga: http://www.java.com/es/download/

- MetaMap. Al comienzo del proyecto la última versión disponible era la 2012, por lo que esta fue la que se utilizó. Durante el periodo de desarrollo se comunicó que pronto podríamos utilizar la versión 2013, sin embargo a fecha de hoy, junio de 2012, aún no disponible.

  Enlace de descarga: http://metamap.nlm.nih.gov/#Downloads.

  Para realizar correctamente dicha instalación, recomendamos al lector dirigirse al anexo Instalación y ejecución de MetaMap y MetaMap Java API, en que se explica más detalladamente los pasos a seguir y los requisitos que deben cumplirse para lograrlo satisfactoriamente dicha instalación será necesario tener una cuenta de usuario de *UMLS*, que se puede solicitar en el siguiente enlace:

  https://uts.nlm.nih.gov//license.html.

  Además cabe destacar que no solo será necesario contar con una versión de MetaMap instalada sino que además será obligatorio haber lanzado previamente los *scripts* correspondientes que, de nuevo, se mencionan en el anexo D.

[1] Esto es únicamente para sistemas *Windows*. Si se trata de otro sistema opertativo, el *JRE* puede ser tanto de 32 como de 64 bits.

A.2. Ejecución y uso del sistema

Durante el resto del capítulo trataremos de describir el modo de uso del sistema. Comenzando por la selección del directorio de trabajo, continuando con el proceso de extracción de información de informes médicos y el buscador de informes y finalizando con menús adicionales como pueden ser el de ajustes o el de ayuda.

A.2.1. Selección del directorio de trabajo

Nada mas iniciar la aplicación, se solicitará al usuario que seleccione la carpeta mediante un aviso como el de la figura A.1, el lugar, en el que desea que se almacenen todos los archivos generados por la aplicación durante la ejecución del programa. Este lugar escogido será el directorio de trabajo actual o *workspace* y contará con tres carpetas, que de no existir, serán creadas:

- pxml/: contiene los archivos `.pxml` generados por el sistema.
- mxml/: contiene los archivos `.mxml` generados por el sistema.
- index/: contiene un conjunto de carpetas que contienen el índice utilizado por *Lucene* (véase apartado 2.9.1).

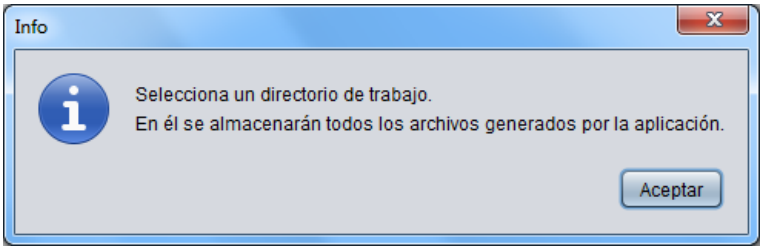


Figura A.1: Solicitud del directorio de trabajo

Así, cuando haya sido escogido, deberemos tener la estructura mostrada en la figura A.2 dentro de la carpeta seleccionada como directorio de trabajo. Resulta fundamental, que el usuario no modifique esta estructura, pues de hacerlo, el programa no funcionará correctamente.

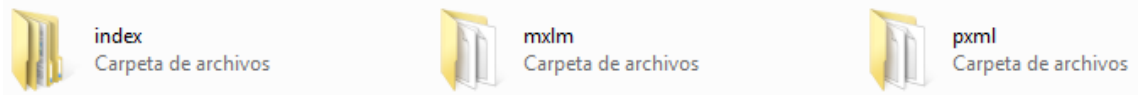


Figura A.2: Estructura del directorio de trabajo

A.2.2. Pantalla principal

Una vez elegido el *workspace*, se presentará una interfaz como la que podemos visualizar en la figura . Esta pantalla nos da la posibilidad de realizar cualquiera de las siguientes acciones:

 Abrir y procesar los informes que poseamos, en el idioma para el que esté configurado el sistema. Tendremos la posibilidad de elegir tanto inglés como español. Puede conocer más sobre este apartado en Abrir y procesar informes.

 Realizar una búsqueda entre los informes que se encuentran en el directorio pxml/ haciendo uso del índice localizado en index/. Esta búsqueda podrá realizarse de diferentes maneras, como podremos ver más adelante en la sección . Para poder realizarla con éxito, será necesario haber creado un índice previamente sobre ese conjunto de informes sobre el que queremos buscar. Para más detalles sobre la creación del índice puede dirigirse a la sección Buscador de informes.

 Ajustar ciertos parámetros del sistema como el idioma de la interfaz, el idioma de los informes a procesar o el el archivo de configuración para el mapeo o la transformación de informes médicos. Para más detalle referenciamos la sección Ajustes.

 Consultar un manual de la aplicación desde Ayuda. Véase Ayuda.

Figura A.3: Menú principal del sistema (español)

A.2.3. Abrir y procesar informes

Si desde la Pantalla principal pulsamos sobre el botón Abrir y Seleccionar Informes aparecerá ante nosotros la ventana que se muestra en la figura A.4. Desde ella podremos ejecutar diferentes acciones. Cuatro de ellas nos permitirán procesar informes de diferen-

tes maneras y otras dos nos darán la posibilidad de cambiar de pantalla como indicamos a continuación.

 Nos permitirá visualizar los informes que se encuentran abiertos por la aplicación. Estos informes se habrán cargado mediante alguna de las opciones que describimos más adelante. Para más detalles sobre esta pantalla consultar Visualizar informes abiertos.

 Nos devolverá a la pantalla principal. Véase Pantalla principal.

Además como hemos mencionado, existen cuatro funciones adicionales que nos permitirán llevar a cabo la actividad propia de la aplicación. Obtener la información de informes médicos y crear una representación a partir de ellos en forma de archivo `.pxml`. Estas cuatro funciones son aplicables tanto a un solo informe, como a todos los que se encuentren en el interior de una misma carpeta en función de la opción que escojamos de las dos siguientes:

- Un único informe
- Todos los informes de una carpeta

Estas cuatro acciones principales que venimos comentando son accesibles mediante el uso de los botones que indicamos a continuación y tienen el cometido que describimos:

- Importar y procesar desde XML: nos da la oportunidad de abrir uno o varios archivos `.xml` y obtener para cada uno de ellos un archivo `.pxml` que representa un informe procesado y que será cargado por la aplicación para que el usuario pueda visualizarlo si así lo desea. Para ello, a partir del archivo `.xml` original se generará un archivo intermedio `.mxml`. De este a su vez será del que finalmente se extraerá la información que servirá para crear el `.pxml`.
- Importar y procesar desde TXT: este botón tiene como función la de realizar las mismas operaciones que el descrito en el punto anterior, salvo que en lugar de llevarlas a cabo tomando como base uno o varios archivos `.xml`, toma como base uno o varios archivos `.txt`.
- Abrir y procesar: en este caso se solicitará al usuario que escoja un único archivo `.mxml` o bien una carpeta en la que se encuentre un conjunto de archivos `.mxml`. De esta manera a partir de cada uno de ellos se generará el archivo `.pxml` correspondiente que será cargado por la aplicación para darnos la oportunidad de visualizarlo si así lo queremos.
- Abrir procesados: este botón invitará al usuario a elegir un archivo `.pxml` o una carpeta en la que se encuentre un conjunto de archivos de este tipo, para que puedan ser cargados por el sistema. Así posteriormente podremos visualizar cada uno de ellos.

Debemos señalar que todos los archivos `.mxml` y `.pxml` generados durante cualquiera de estos procesos, serán almacenados en el *workspace* actual que tenga asociado el sistema.

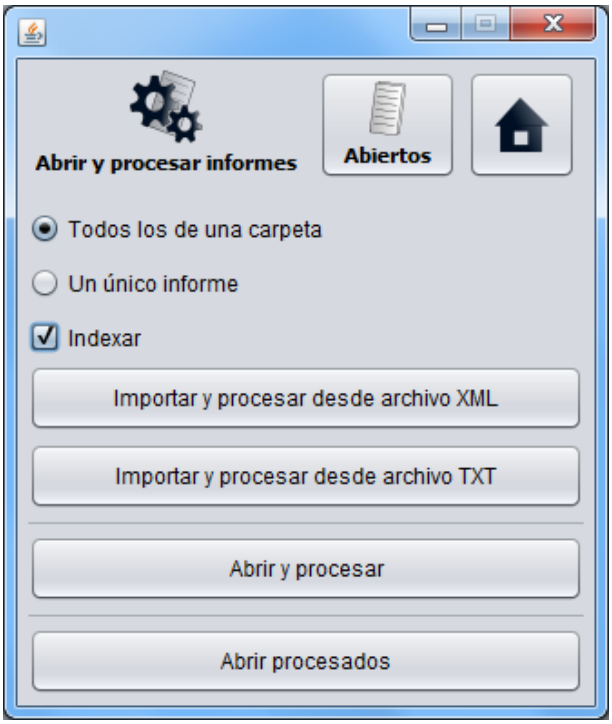


Figura A.4: Pantalla del sistema Abrir y procesar informes

A.2.3.1. Indexar informes

Por otro lado podemos ver en la figura A.4 un *checkbox* rotulado como Indexar que el usuario puede activar si lo desea, siempre que se haya escogido la opción Todos los de una carpeta. La activación de este *checkbox* tiene como propósito indicar a la aplicación que deseamos que se realice un indexado al final de la ejecución. Esto es, después de completarse la acción seleccionada de entre las principales posibles, que se sobrescriba el índice contenido en la carpeta index/ del *workpspace* por un índice nuevo. Dicho índice será generado a partir de todos los archivos que se encuentren en el directorio pxml/ del propio *workspace* (incluyendo por tanto no solo los que se acaban de generar, sino también los que allí se encontraban).

A.2.3.2. Visualizar informes abiertos

Una vez que hemos ejecutado alguna de las opciones principales que nos permite el menú Abrir y procesar informes, se nos mostrará, siempre que los archivos seleccionado sean válidos, la pantalla de la figura A.5.

Esta pantalla nos presenta una tabla con todos los informes abiertos por la aplicación y a través de la selección de uno de ellos y del uso del botón Ver informe, podremos acceder a la visualización de un informe individual.

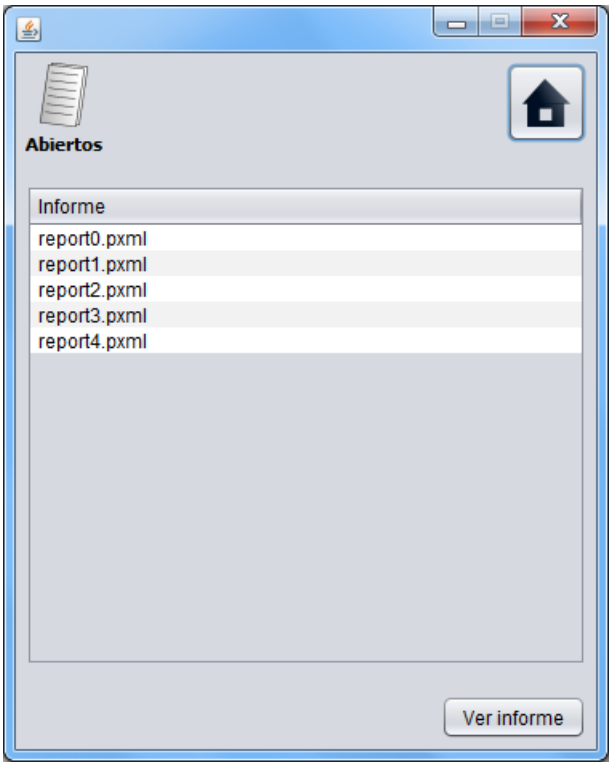


Figura A.5: Pantalla del sistema: Informes Abiertos

Esta visualización puede presentársenos de dos maneras en función de cuál haya sido la última forma de cargar informes en el sistema que hayamos utilizado.

Vista simple: `pxml`

En caso de la última operación ejecutada, del menú Abrir y procesar informes, haya sido Abrir informes procesados, entonces encontraremos ante nosotros algo similar a la figura A.6. Lo que vemos es el contenido del archivo `.pxml` seleccionado en la tabla. Este contenido muestra todos aquellos conceptos médicos que han sido detectados en cada una de las secciones del informe tras realizar el procesamiento del mismo. Cada concepto, tal como vemos puede estar dentro del conjunto Conceptos afirmados, Conceptos negados o Conceptos especulados. Esto hace referencia a tipo de frase en la que se ha detectado dicho concepto. Si teníamos algo similar a "El paciente ha llegado al hospital con un esguince" entonces el concepto esguince aparecerá en Conceptos afirmados. En caso de que tuviésemos algo como "El paciente no tiene un esguince" lo encontraemos en Conceptos negados. Por último si la frase original decía algo como "El paciente tal vez tenga un esguince; se le realizarán pruebas para verificarlo" podremos encontrar el concepto médico esguince dentro de Conceptos especulados.

Vista doble: `mxml` **y** `pxml`

Por otro lado si la última operación llevada a cabo, del menú Abrir y procesar informes, *no* ha sido Abrir informes procesados, entonces se nos mostrará una pantalla con una vista

doble. Por un lado la parte derecha presentará el contenido del archivo `.pxml` seleccionado tal como se ha descrito en el punto anterior. Por otro, la parte derecha mostrará el contenido del archivo .mxml asociado a ese `.pxml`. Dicho contenido consistirá en el texto presente en cada una de las secciones en que se divide un archivo `.mxml`. Esto es lo que vemos en la figura A.7. Para saber más sobre este tipo de archivos puede consultar la sección 3.2.1.

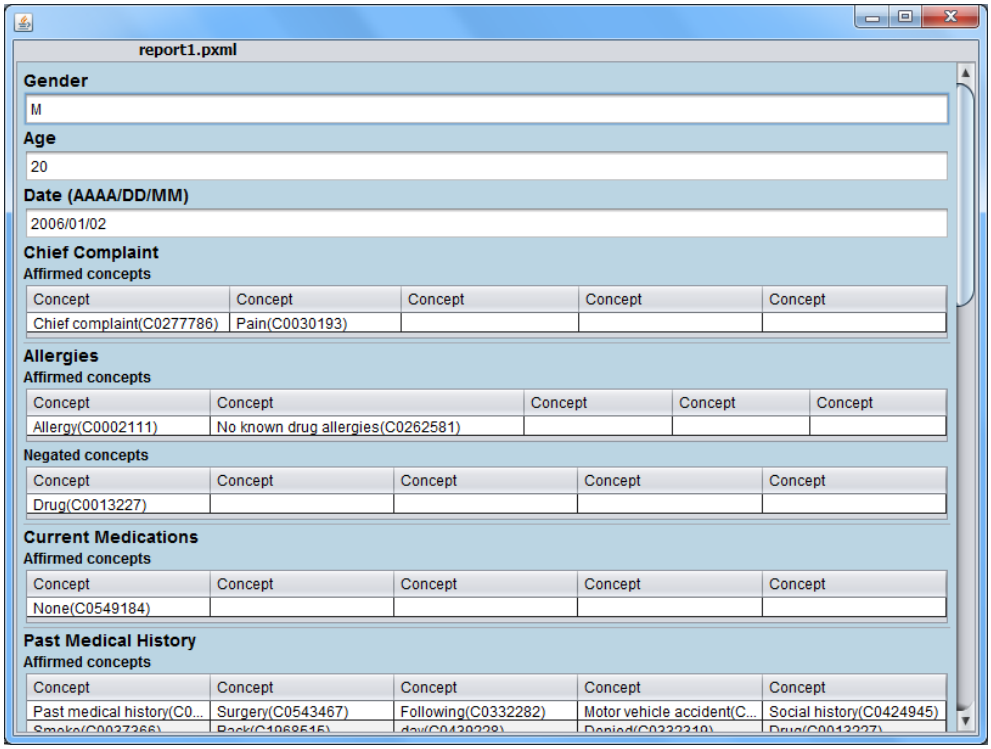


Figura A.6: Visualizador de informes médicos. Vista simple: `pxml`

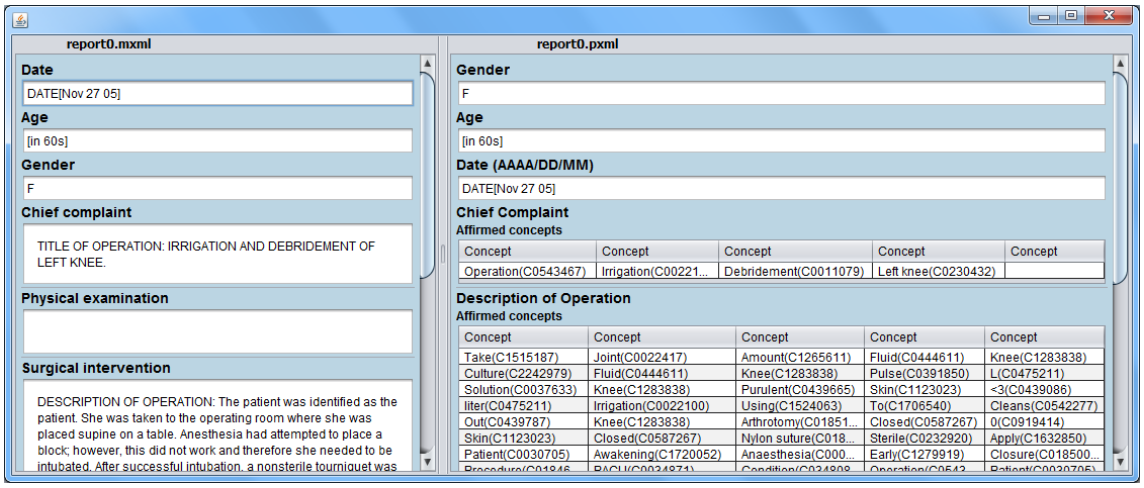


Figura A.7: Visualizador de informes médicos. Vista doble: `mxml` y `pxml`

A.2.4. Buscador de informes

Si, estando en la Pantalla principal, pulsamos sobre el botón Buscador de informes veremos la ventana incluida en la figura A.8. Desde ella el usuario puede realizar búsquedas de informes médicos en formato `.pxml` indexados previamente (véase apartado del manual Indexar informes). Dichas búsquedas podrán realizarse de dos maneras en función de cuál sea el botón pulsado.

1. Seleccionar informe para buscar otros similares: en caso de que sea este el botón escogido, se mostrará al usuario un diálogo para que elija un archivo .pxml a través del cual, pueda el sistema, encontrar otros informes similares (también en formato .pxml).

2. Buscar: para conseguir realizar con éxito búsquedas a través de este método, la persona que esté ejecutando el sistema deberá completar, antes de presionar este botón, aquellas partes que desee del formulario. Así indicará cuáles son las características que le interesa que cumplan aquellos informes médicos que quiere encontrar. Para rellenar el formulario, no tendrá más que indicar qué sexo debe tener el paciente del informe, en qué rango de edad se encuentra dicho paciente y entre qué fechas mínima y máxima (incluidas) se redactó el informe. En cuanto al resto del formulario bastará con detallar qué palabras clave (conceptos médicos) se desea que estén presentes en una determinada sección y en qué estado (afirmado, negado o especulado). Así, tal como mencionábamos en el apartado A.2.3.2, si dentro una sección concreta (Motivo de ingreso, Juicio clínico principal etc.), deseamos encontrar algo como "El paciente ha llegado al hospital con un esguince" entonces el concepto esguince deberemos incluirlo en Conceptos afirmados. En caso de que queramos encontrar algo como "El paciente no tiene un esguince" escribiremos esguince en Conceptos negados. Y por último si buscamos una frase similar a "El paciente tal vez tenga un esguince; se le realizarán pruebas para verificarlo" deberemos incluir el concepto médico esguince dentro de Conceptos especulados. Si deseamos filtrar los resultados por el sexo del paciente, deberemos indicarlo en el campo correspondiente. Si deseamos filtrar por un rango de edad, se deberá poner en el campo adecuado (si se dejan los valores a 0, no se realizará el filtrado por la edad) y finalmente, si se desea filtrar por fecha del informe, se deberá indicar en el campo correspondientes con el formato AAAA/D-D/MM. Si en alguna búsqueda no se incluye nada en los campos de conceptos, la búsqueda no devolverá resultado alguno.

Independientemente de cuál sea el método de búsqueda elegido, esta se realizará utilizando el último índice contenido en la carpeta index/ del *workspace,* que a su vez habrá sido creado tomando como base los archivos que había en ese momento en el directorio pxml/. Además cabe señalar que como máximo el número de resultados devueltos por la consulta será aquel indicado mediante el campo etiquetado como Núm. máx. de resultados.

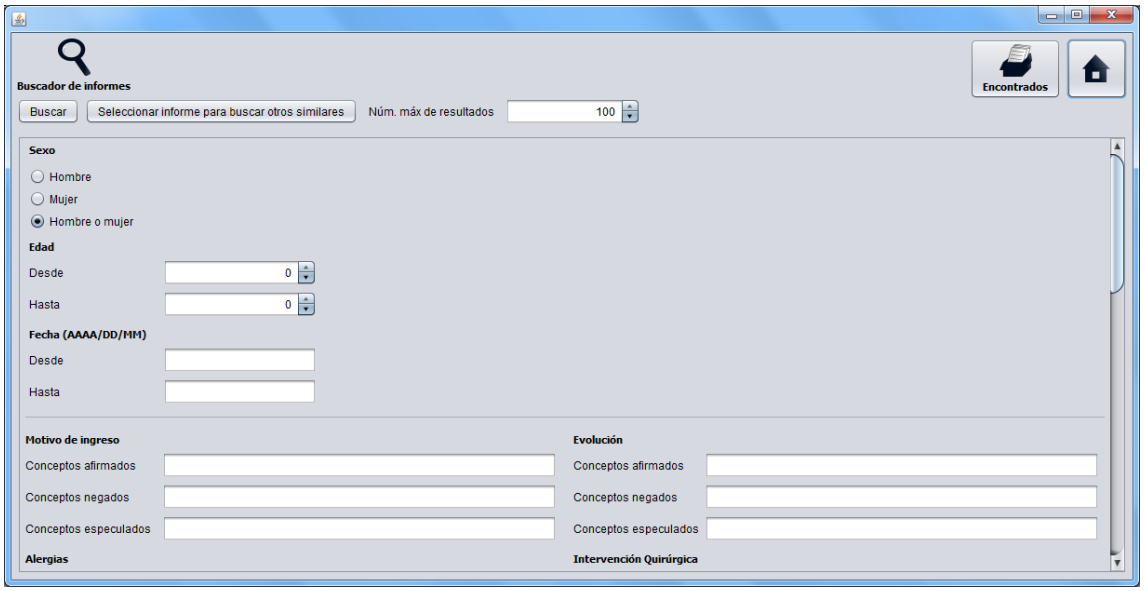


Figura A.8: Pantalla del sistema Buscador de informes

A.2.4.1. Visualizar los informes obtenidos tras la búsqueda

Una vez que se ha hecho uso del botón correspondiente, se realizará la búsqueda. En caso de que no hubiese resultados, se indicaría con un aviso como el de la figura A.10, y en caso de que el sistema sí hubiese localizado un conjunto de informes que cumpliese los requisitos de búsqueda, se nos mostrará un mensaje indicándonos cuántos resultados se han localizado (figura A.9) y a continuación una pantalla con todos ellos como la que vemos en la figura A.11. Desde esta última pantalla tendremos la opción de abrirlos y consultarlos de manera equivalente a como describíamos en el apartado Vista simple: `pxml`.

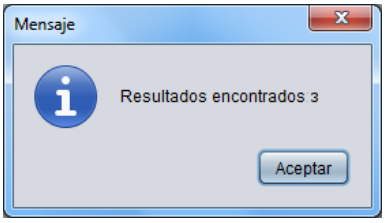


Figura A.9: Búsqueda con resultados

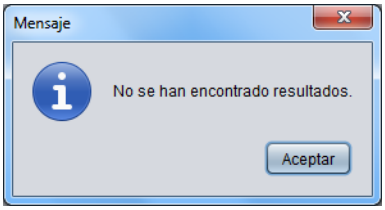


Figura A.10: Búsqueda sin resultados

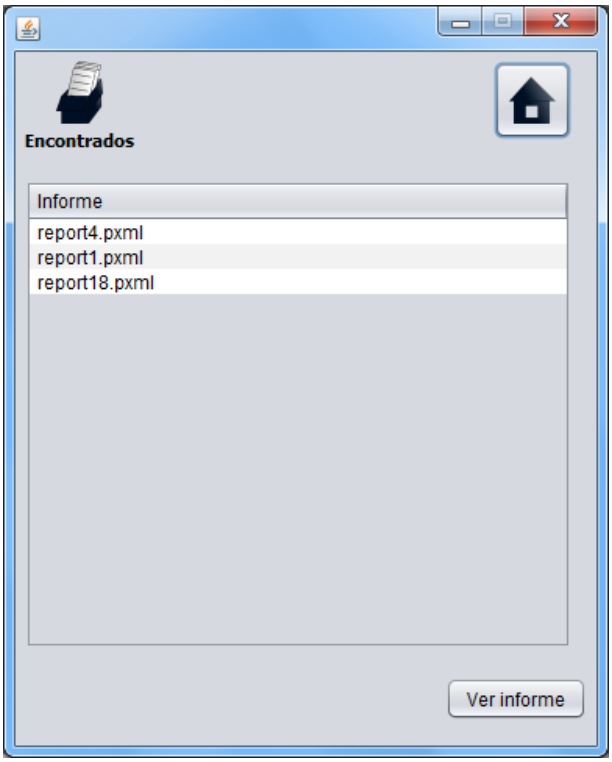


Figura A.11: Pantalla del sistema Informes Encontrados

A.2.5. Ajustes

Otra de las pantallas con que cuenta el sistema, es la de ajustes donde podremos elegir el idioma de la interfaz gráfica, indicar el idioma en que estarán los informes médicos que vayamos a tratar y editar los archivos de configuración para transformación o mapeo de informes médicos desde archivos con formato `.xml` o `.txt`.

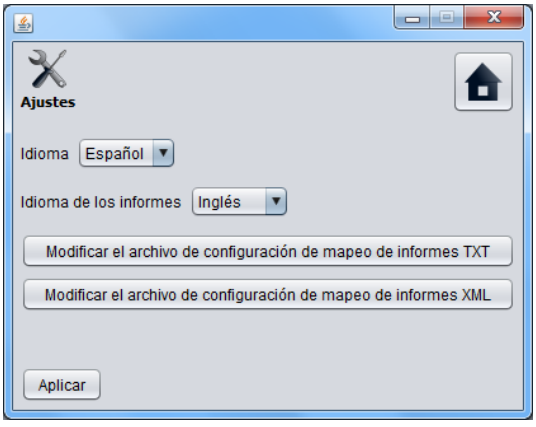


Figura A.12: Pantalla del sistema Ajustes

Una vez realizados los cambios, y tras pulsar el botón Aplicar, podremos comprobar cómo los parámetros se han configurado según nuestras elecciones e indicaciones. El ejemplo

más visual, por supuesto, es el cambio de idioma de la interfaz. Al comienzo de este manual, en la figura A.3 podíamos visualizar la pantalla principal del sistema en español. En la figura A.13, se nos muestra la misma ventana principal tras haber utilizado el menú de ajustes y haber modificado el idioma de la interfaz para que pase a ser inglés.

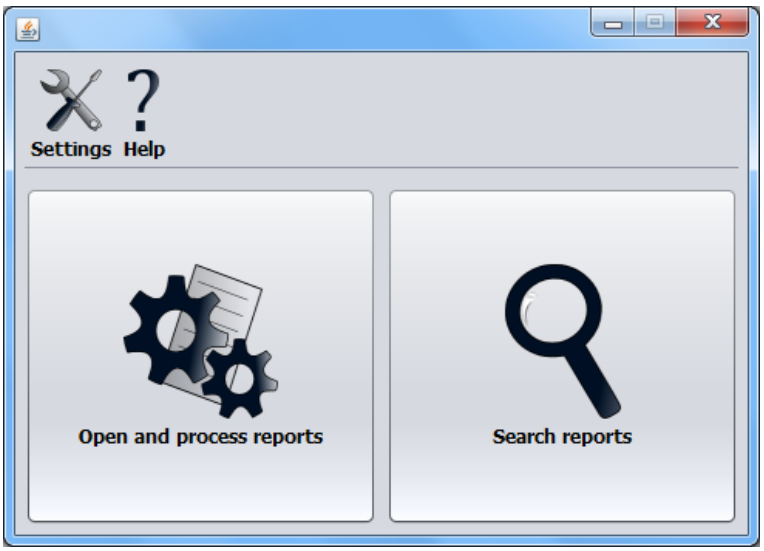


Figura A.13: Menú principal del sistema (inglés)

A.2.5.1. Modificar el archivo de configuración de mapeo de informes `.txt`

Si pulsamos el botón de modificar el archivo de configuración de mapeo de informes `.txt` nos aparecerá una nueva ventana como la que aparece en la figura A.14. En este formulario tendremos que introducir para cada campo por que cadena de palabras empieza en el `.txt`, si alguno de los campos no estuviese en el `.txt` habría que dejar dicho campo en blanco. Si un campo puede empezar por más de una cadena de palabras diferentes habrá que poner las distintas cadenas por las que pueda empezar separadas por comas. Además hay que rellenar el campo de palabras de parada en el que hay que incluir todas las cadenas de palabras que indiquen que un campo ha terminado, de la misma forma que antes si pueden ser varias cadenas habrá que ponerlas todas separadas por comas.

A.2.5.2. Modificar el archivo de configuración de mapeo de informes `.xml`

Si pulsamos el botón de modificar el archivo de configuración de mapeo de informes `.xml` nos aparecerá una nueva ventana como la que aparece en la figura A.15. En este formulario tendremos que indicar para cada campo cuál es la etiqueta equivalente en el `.xml` de origen, si alguno de los campos no estuviese en el `.xml` de origen bastaría con dejarlo en blanco.

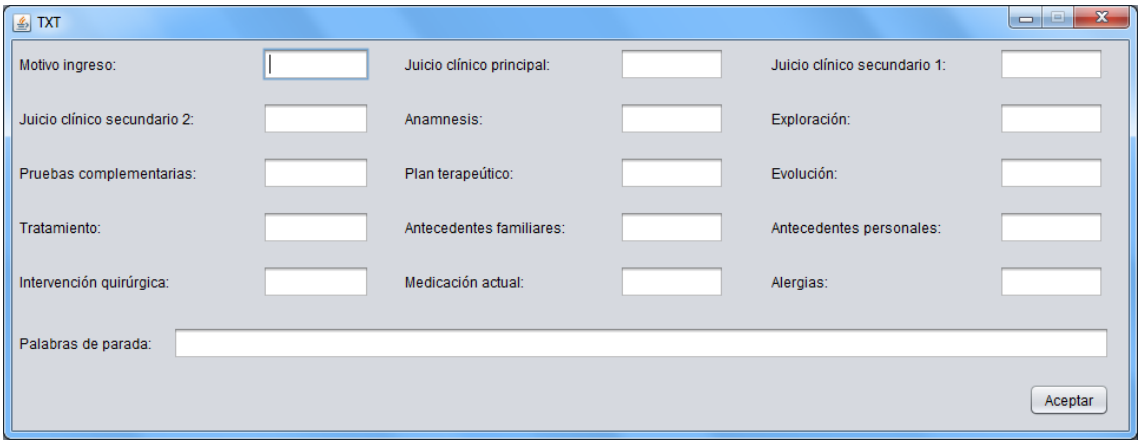


Figura A.14: Formulario para modificar el archivo de configuración para mapeo de `txt`

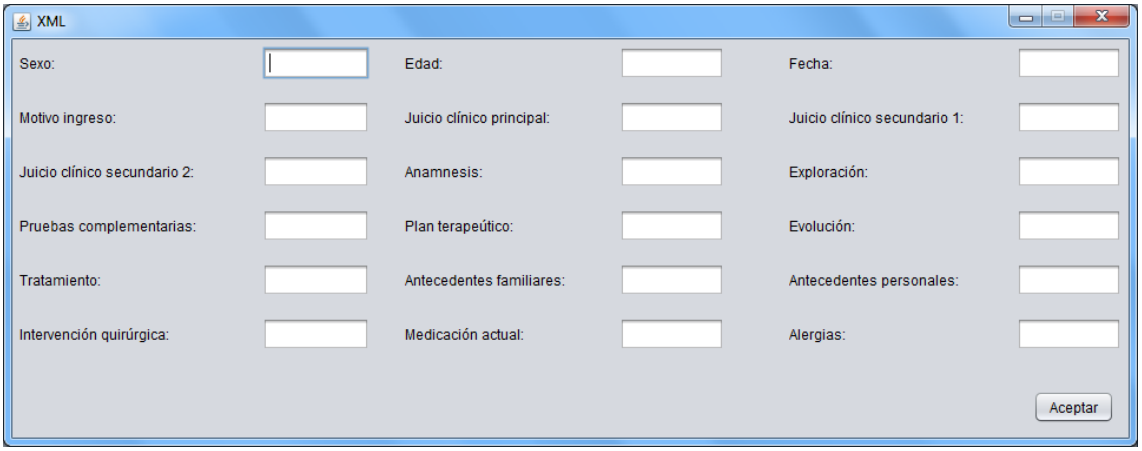


Figura A.15: Formulario para modificar el archivo de configuración para mapeo de `xml`

A.2.6. Ayuda

Por si el usuario necesita ayuda durante la ejecución, se ha incluido un botón en la aplicación que abrirá una página web en aquel navegador que esté configurado como predeterminado por el usuario (figura A.16). En ella se explica cómo utilizar las características más importantes del sistema. El individuo que esté haciendo uso de ella, no tendrá más que navegar por los diferentes apartados de la página para localizar el apartado buscado y resolver las dudas que pudiese tener a la hora de ejecutar una determinada acción.

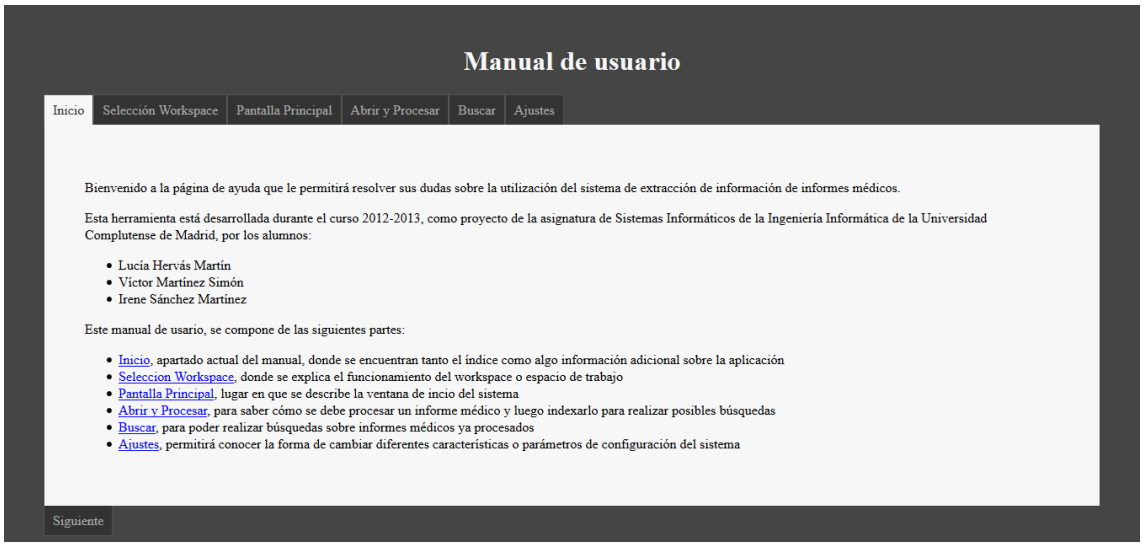


Figura A.16: Página web de ayuda del sistema

Apéndice B

Configuración

Para modificar ciertos parámetros de ejecución, se han creado diferentes archivos de configuración. que se encuentran en la carpeta `conf/` dentro del directorio principal de la aplicación. Estos archivos podrán ser modificados por el usuario a través de cualquier editor de texto. Algunos de ellos, además, dada su importancia, podrán modificarse desde la propia interfaz del sistema. Ciertos archivos de configuración serán comunes para los dos idiomas, mientras que otros estarán duplicados, por estar disponibles tanto para inglés como para español. A continuación indicaremos cuál es el contenido de cada uno de ellos y el directorio en que se encuentran.

B.1. `conf/Common/`

En esta ruta se encuentran todos aquellos archivos configuración que son comunes a ambos idiomas.

B.1.1. `language.properties`

Este archivo permite especificar el idioma tanto de la interfaz a la hora de ejecutar la aplicación, como el idioma en que se encuentran los archivos a procesar. Es decir, señala cuál es el procesador que se debe utilizar a la hora de tratar los informes médicos, el que permite extraer su información suponiendo que están en inglés o el que permite hacerlo dando por hecho que están en español. Dada su importancia, el contenido de este archivo de configuración puede modificarse desde el panel de Ajustes de la propia aplicación.

```
# Set LANGUAGE property to modify the application language.
# It must be one of these values {spa, eng}
# Default: spa
# Set REPORTS_LANGUAGE property to specify the
# language of the reports to process.
# It must be one of these values {spa, eng}
# Default: eng
REPORTS_LANGUAGE=eng
LANGUAGE=eng
```

Listado de código B.1: `language.properties`

B.1.2. `txt.properties`

Este documento, que de nuevo podemos modificar desde el panel de Ajustes, almacena la información necesaria para mapear informes médicos desde archivos `.txt` y transformarlos en archivos `.mxml`. De esta manera, vemos que contiene para cada una de las secciones con que cuenta un informe según nuestra definición (véase apartado 3.2), una lista de conjuntos de palabras, que permitirán identificar dicha sección dentro de un texto plano.

```
#####################################################
# Fichero de configuracion de los informes de texto #
#####################################################
motivo_ingreso=REASON FOR CONSULTATION,TITLE OF OPERATION,CHIEF
    COMPLAINT,CLINICAL
jcprincipal=DIAGNOSIS,DIAGNOSTIC,DISCHARGE DIAGNOSIS,ADMISSION
    DIAGNOSIS,ADMITTING DIAGNOS,FINAL DIAGNOSIS
jcsecundario1=DIFFERENTIAL
jcsecundario2=
anamnesis=HISTORY OF PRESENT ILLNESS
exploracion=REVIEW OF SYSTEMS,PHYSICAL EXAM,FINDINGS,HISTORY AND
    INDICATIONS, REVIEW OF SYMPTOMS
pruebas_complementarias=LAB,CHEST X-RAY,IMAGING,RADIOGRAPHIC
plan_terapeutico=IMPRESSION,PLAN,DISPOSITION,HOSPITAL COURSE
evolucion=ED COURSE,ASSESSMENT
tratamiento=TREATMENT,DISCHARGE INSTRUCTIONS,FOLLOW-UP INSTRUCTIONS
antecedentes_familiares=FAMILY HISTORY,SOCIAL AND FAMILY HISTORY
antecedentes_personales=PAST MEDICAL,PRIOR MEDICAL,SOCIAL HISTORY
intervencion_quirurgica=DESCRIPTION OF OPERATION
medicacion_actual=MEDICATIONS,CURRENT MEDICATIONS,DISCHARGE
    MEDICATIONS
alergias=ALLERGIES
parada=ANESTHESIA:,DISCHARGE,PROCEDURE(S),GOALS,FOLLOW-UP
    INSTRUCTIONS,FINDINGS,END,CLINICAL,HOSPITAL,EMERGENCY,
    LABORATORIES,REASON,TITLE,CHIEF,HISTORY OF PRESENT ILLNESS,PAST,
    PRIOR,SOCIAL,FAMILY,ALLERGIES,MEDICATIONS,CURRENT,REVIEW,
    PHYSICAL,LABORATORY,LAB,LABS,CHEST X-RAY,IMAGING,RADIOGRAPHIC,
    DESCRIPTION,ED,DIAGNOSIS,ADMISSION,DIAGNOSTIC,DIFFERENTIAL,
    IMPRESSION,ASSESSMENT,PLAN,DISPOSITION,ASSESSMENT/PLAN,ATTENDING
```

Listado de código B.2: `txt.properties`

B.1.3. `xml.properties`

Este documento, similar al anterior, y que supone el último de los que pueden editarse desde el panel de Ajustes, contiene la información necesaria para mapear informes médicos desde archivos `.xml` y transformarlos en archivos `.mxml`. Así, para cada una de las

secciones con que cuenta un informe según nuestra definición (véase apartado 3.2), aparece una etiqueta relativa que pertenecerá al `.xml` original y que permitirán identificar dicha sección dentro de un texto plano.

```
#############################################################
# Fichero de configuracion de la estructura de los informes #
#############################################################
sexo=sexo
edad=no
fecha=fecha_publicacion
motivo_ingreso=motivo_ingreso
jcprincipal=jcprincipal
jcsecundario1=jcsecundario1
jcsecundario2=jcsecundario2
anamnesis=anamnesis
exploracion=exploracion
pruebas_complementarias=pruebas_complementarias
plan_terapeutico=plan_terapeutico
evolucion=evolucion
tratamiento=tratamiento
antecedentes_familiares=familiares
antecedentes_personales=personales
intervencion_quirurgica=intervencion_quirurgica
medicacion_actual=no
alergias=no
```

Listado de código B.3: `xml.properties`

B.2. conf/Spanish

Los archivos almacenados en este directorio, son aquellos que nos van a permitir conocer la configuración de los diferentes módulos de procesamiento de informes médicos en español.

B.2.1. splitter.properties

Este documento permitirá al sistema conocer las diferentes rutas en las que se encuentran los distintos archivos que darán la posibilidad de dividir el texto para que pueda ser tratado.

```
#########################################################
# Fichero de configuracion del splitter de frases
#########################################################
# Codificacion de los ficheros de patrones del splitter
SPLITTER_ENCODING=UTF-8
# Ruta de los patrones del splitter
RUTA_REGEXP =./resources/Spanish/sentSplitterRE
# Patrones de separacion de parrafos o bloques de texto
EXTERNAL_SPLIT_LIST=external-split-patterns.txt
```

```
# Patrones de separacion de frases de un parrafo
INTERNAL_SPLIT_LIST=internal-split-patterns.txt
# Ampliacion de los anteriores patrones para obtener
# subfrases en la fase de conceptos
INTERNAL_SPLIT_CONCEPTOS=internal-split-concep.txt
# Patrones que podrian parecer splits, pero no lo son
NON_SPLIT_LIST=non-split-patterns.txt
```

Listado de código B.4: `splitter.properties`

B.2.2. `correccion.properties`

En este documento podemos encontrar la información relativa a la fase de corrección ortográfica. Esta información aporta datos como la localización de la librería que permite utilizar *Hunspell*, la situación del diccionario que contiene los términos que se darán por correctos, el peso asociado a cada uno de los métodos de comparación de cadenas que permitirán puntuar sugerencias, o por último la puntuación mínima que debe lograr una sugerencia para que sea dada por correcta.

```
#######################################################
# Fichero de configuracion de la fase de correccion #
#######################################################
# Ruta de la libreria Hunspell
LIB_HUNSPELL=lib/CorrAcr/hunspell/
# Directorio de los diccionarios
HOMEDICC=resources/Spanish/Diccionario/
# Nombre del diccionario a utilizar (sin la extension)
DICCIONARIO=es-ES
# Numero maximo de sugerencias del corrector
NUMERO_SUGERENCIAS=5
# Peso distancia Levenshtein
PESO_LEVENSHTEIN=0.33
# Peso distancia Needleman Wunsch
PESO_NEEDLEMAN=0.33
# Peso distancia fonetica
PESO_FONETICA=0.33
# Umbral que debe sobrepasar la sugerencia final
UMBRAL=0.9
```

Listado de código B.5: `correccion.properties`

B.2.3. `acronimos.properties`

A lo largo de este documento se detallan todos los datos que permiten ejecutar con éxito la fase de expansión de acrónimos en español. Así podemos ver propiedades que indican la localización de las bases de datos o que señalan si se hará uso o no de la desambiguación.

```
#########################################################
# Fichero de configuracion de la fase de acronimos
```

```
3  ########################################################
4  # Recurso de acronimos que se quiere utilizar (xml, bbdd o thes)
5  SELECCION_RECURSO=bbdd
6  # Archivo XML de expansiones de acronimos
7  ACRONIMOS_XML=./resources/Spanish/Acronimos.xml
8  # Directorio de la base de datos
9  URL_BBDD=resources/Spanish/BBDD
10 # Usuario de la base de datos de SNOMED
11 USUARIO_BBDD=usuario
12 # Clave de la base de datos de SNOMED
13 CLAVE_BBDD=1234
14 # Fichero de acronimos que no hay que expandir
15 URL_EXCEPCIONES=resources/Spanish/excepcionesAcronimos
16 # Tabla de descripciones de conceptos
17 NOMBRE_BBDD_SNOMEDCT=snomed.db
18 # Base de datos de acronimos
19 NOMBRE_BBDD_ACRONIMOS=acronimos.db
20 # Base de datos de reglas
21 NOMBRE_BBDD_REGLAS=acronimos.db
22 # Tabla de reglas
23 NOMBRE_TABLA_REGLAS=acronim_rules
24 # Nombre de la tabla de descripciones
25 NOMBRE_TABLA_DESCRIP=t_descripciones_snomed
26 # Nombre de la tabla de acronimos
27 NOMBRE_TABLA_LISTA_ACRONIMOS=acronim_list
28 # Nombre de la tabla de expansiones
29 NOMBRE_TABLA_EXPANSION_ACRONIMOS=acronim_description
30 # Tabla de acronimos
31 NOMBRE_BBDD_THESAURO=acronimos.db
32 # Nombre de la tabla de descripciones
33 NOMBRE_TABLA_THESAURO=acronim_description
34 # Nombre de la tabla de features de los acronimos
35 NOMBRE_TABLA_FEATURES=features
36 # Indica si se utilizan las reglas de expansion
37 USO_REGLAS=true
38 # Indica si se utiliza la comparacion entre contextos para la
      desambiguacion
39 USO_CONTEXTOS=true
40 # Tipo de la base de datos: mysql o sqlite
41 TIPO_BBDD=sqlite
```

Listado de código B.6: `acronimos.properties`

B.2.4. `negacion.properties`

En este documento se detallan los archivos de texto que se utilizan para poder detectar las negaciones correctamente mediante el algoritmo NegEx.

```
1  ########################################################
2  # Fichero de configuracion de la fase de negacion
3  ########################################################
4  # Ruta de los triggers de NegEx
```

```
RUTA_TRIGGERS_NEGEX =./resources/Spanish/negExTriggers
# Fichero de triggers de negacion
NOMBRE_FICHERO_NEG=negPhrases.txt
# Fichero de triggers de pseudo-negacion
NOMBRE_FICHERO_PSENEG=pseNegPhrases.txt
# Fichero de triggers de post-negacion
NOMBRE_FICHERO_POSTNEG=postNegPhrases.txt
# Fichero de triggers de conjunciones
NOMBRE_FICHERO_CONJ=conjunctions.txt
```

Listado de código B.7: `negacion.properties`

B.2.5. `condicional.properties`

En el siguiente documento nos encontraremos con la información relativa a los archivos de texto utilizados para poder detectar frases especuladas.

```
##########################################################
# Fichero de configuracion de la fase de negacion
##########################################################
# Ruta de los triggers de concionales
RUTA_FICHEROS =./resources/Spanish/condicionales
# Fichero de triggers de auxiliares
NOMBRE_FICHERO_AUX=auxiliares.txt
# Fichero de triggers de verbos
NOMBRE_FICHERO_VERBOS=verbos.txt
# Fichero de triggers de adjetivos
NOMBRE_FICHERO_ADJETIVOS=adjetivos.txt
# Fichero de triggers de conjunciones
NOMBRE_FICHERO_CONJUNCIONES=conjunciones.txt
# Fichero de triggers de adverbios
NOMBRE_FICHERO_ADVERBIOS=adverbios.txt
```

Listado de código B.8: `condicional.properties`

B.2.6. `metamap.properties`

Mediante este archivo configuramos el uso de MetaMap. Aunque actualmente este archivo tenga los mismos datos en las carpetas `English` y `Spanish` lo tenemos por duplicado porque nos permite mayor independencia entre los procesadores y además nos permite lanzar simultáneamente dos servidores de MetaMap, lo cual podría interesar en un momento dado.

```
##########################################################
# Fichero de configuracion de la fase de negacion
##########################################################
# Puerto donde se esta ejecutando metamap
PORT=8066
# Direccion donde se esta ejecutando metamap
```

```
HOST=localhost
# time out de metamap
TIMEOUT=0
```

Listado de código B.9: metamap.properties

B.3. conf/English

Los archivos de configuración de esta carpeta son homólogos a los archivos con el mismo nombre de la carpeta Spanish. El archivo de configuración para las negaciones no es necesario debido a que en inglés se utiliza MetaMap para las negaciones. Aunque para los acrónimos también utilizamos MetaMap y el archivo de configuración correspondiente podría borrarse, hemos decidido dejarlo por si en algún momento se quisiera volver a utilizar una base de datos para expandir los acrónimos.

B.3.1. splitter.properties

```
#############################################################
# Fichero de configuracion del splitter de frases
#############################################################
# Codificacion de los ficheros de patrones del splitter
SPLITTER_ENCODING=UTF-8
# Ruta de los patrones del splitter
RUTA_REGEXP =./resources/Spanish/sentSplitterRE
# Patrones de separacion de parrafos o bloques de texto
EXTERNAL_SPLIT_LIST=external-split-patterns.txt
# Patrones de separacion de frases de un parrafo
INTERNAL_SPLIT_LIST=internal-split-patterns.txt
# Ampliacion de los anteriores patrones para obtener
# subfrases en la fase de conceptos
INTERNAL_SPLIT_CONCEPTOS=internal-split-concep.txt
# Patrones que podrian parecer splits, pero no lo son
NON_SPLIT_LIST=non-split-patterns.txt
```

Listado de código B.10: splitter.properties

B.3.2. correccion.properties

```
#############################################################
# Fichero de configuracion de la fase de correccion
#############################################################
# Ruta de la libreria Hunspell
LIB_HUNSPELL=lib/CorrAcr/hunspell/
# Directorio de los diccionarios
HOMEDICC=resources/Enlish/Diccionario/
# Nombre del diccionario a utilizar (sin la extension)
```

```
DICCIONARIO=my_en_med_out_completo
# Numero maximo de sugerencias del corrector
NUMERO_SUGERENCIAS=5
# Peso distancia Levenshtein
PESO_LEVENSHTEIN=0.33
# Peso distancia Needleman Wunsch
PESO_NEEDLEMAN=0.33
# Peso distancia fonetica
PESO_FONETICA=0.33
# Umbral que debe sobrepasar la sugerencia final
UMBRAL=0.9
```

Listado de código B.11: `correccion.properties`

B.3.3. `acronimos.properties`

```
############################################################
# Fichero de configuracion de la fase de acronimos
############################################################
# Recurso de acronimos que se quiere utilizar (xml, bbdd o thes)
SELECCION_RECURSO=bbdd
# Archivo XML de expansiones de acronimos
ACRONIMOS_XML=./resources/English/Acronimos.xml
# Directorio de la base de datos
URL_BBDD=resources/English/BBDD
# Usuario de la base de datos de SNOMED
USUARIO_BBDD=usuario
# Clave de la base de datos de SNOMED
CLAVE_BBDD=1234
# Fichero de acronimos que no hay que expandir
URL_EXCEPCIONES=resources/English/excepcionesAcronimos
# Tabla de descripciones de conceptos
NOMBRE_BBDD_SNOMEDCT=snomed.db
# Base de datos de acronimos
NOMBRE_BBDD_ACRONIMOS=acronimos.db
# Base de datos de reglas
NOMBRE_BBDD_REGLAS=acronimos.db
# Tabla de reglas
NOMBRE_TABLA_REGLAS=acronim_rules
# Nombre de la tabla de descripciones
NOMBRE_TABLA_DESCRIP=descripciones_snomed
# Nombre de la tabla de acronimos
NOMBRE_TABLA_LISTA_ACRONIMOS=acronim_list
# Nombre de la tabla de expansiones
NOMBRE_TABLA_EXPANSION_ACRONIMOS=acronim_description
# Tabla de acronimos
NOMBRE_BBDD_THESAURO=acronimos.db
# Nombre de la tabla de descripciones
NOMBRE_TABLA_THESAURO=acronim_description
# Nombre de la tabla de features de los acronimos
NOMBRE_TABLA_FEATURES=features
# Indica si se utilizan las reglas de expansion
```

```
37 USO_REGLAS=true
38 # Indica si se utiliza la comparacion entre contextos para la
       desambiguacion
39 USO_CONTEXTOS=true
40 # Tipo de la base de datos: mysql o sqlite
41 TIPO_BBDD=sqlite
```

Listado de código B.12: acronimos.properties

B.3.4. condicional.properties

```
1  ############################################################
2  # Fichero de configuracion de la fase de negacion
3  ############################################################
4  # Ruta de los triggers de concionales
5  RUTA_FICHEROS =./resources/English/condicionales
6  # Fichero de triggers de auxiliares
7  NOMBRE_FICHERO_AUX=auxiliares.txt
8  # Fichero de triggers de verbos
9  NOMBRE_FICHERO_VERBOS=verbos.txt
10 # Fichero de triggers de adjetivos
11 NOMBRE_FICHERO_ADJETIVOS=adjetivos.txt
12 # Fichero de triggers de conjunciones
13 NOMBRE_FICHERO_CONJUNCIONES=conjunciones.txt
14 # Fichero de triggers de adverbios
15 NOMBRE_FICHERO_ADVERBIOS=adverbios.txt
```

Listado de código B.13: condicional.properties

B.3.5. metamap.properties

```
1  ############################################################
2  # Fichero de configuracion de la fase de negacion
3  ############################################################
4  # Puerto donde se esta ejecutando metamap
5  PORT=8066
6  # Direccion donde se esta ejecutando metamap
7  HOST=localhost
8  # time out de metamap
9  TIMEOUT=0
```

Listado de código B.14: metamap.properties

Apéndice C

Creación de una ontología personalizada

Tal como se mencionó en la sección 2.2.2.2, *UMLS* contiene tal cantidad de fuentes de vocabulario que en rara ocasión es necesario utilizarlas todas. En nuestro caso, no solo por eso si porque además dicho exceso ocasionaba un aumento de errores por ruido introducido, decidimos crear una base de datos personalizada para ser usada con MetaMap.

Para alcanzar este objetivo fue necesario realizar los diferentes pasos que se explican en los siguientes apartados.

C.1. Requisitos previos

Antes de poder realizar la ontología personalizada, debemos:

1. Contar con un ordenador que funcione sobre una distribución de GNU-Linux.
2. Tener instalada una distribución *UMLS*, así como una de MetamorphoSys.
3. Tener instalado el *LVG*[1].
4. Tener instalado el *DataFileBuilder*[2] teniendo en cuenta las indicaciones correspondientes.[3].

C.2. Creando la base con MetamorphoSys

A continuación es el momento de crear las bases de datos que usaremos como base de toda la ontología, para ello:

1. Arrancamos MetamorphoSys moviéndonos a la ruta en que está descomprimida la versión de *UMLS* y ejecutamos el *script* `run_linux.sh`.

[1] http://lexsrv3.nlm.nih.gov/LexSysGroup/Projects/lvg/current/web/download.html
[2] http://metamap.nlm.nih.gov/#DatafileBuilder
[3] http://metamap.nlm.nih.gov/README_dfb.html

2. Seleccionamos *Install UMLS*.

3. Seleccionamos la rutas de destino y de origen de los datos y pulsamos *OK*.

4. Creamos una nueva configuración con las características que deseemos (fuentes, conceptos aceptados...).

5. La única condición obligatoria consiste en establecer, en la pestaña *Output Options*, *ORF* como formato de salida.

6. Finalmente cuando hayamos completado la configuración, seleccionaremos la opción *Done* en el menú, al hacerlo se comenzará con la generación de los archivos base.

7. Una vez finalizado, debemos asegurarnos que los siguientes archivos están generados:

   a) En la carpeta `META`:

      1) `MRCON`
      2) `MRSTAT`
      3) `MRSO`
      4) `MRSTY`
      5) `MRSAB`
      6) `MRRANK`

   b) En la carpeta `LEX/LEX_DB`:

      1) `SM.DB`

C.3. Creación del WorkSpace

Para una correcta instalación, se hará todo el trabajo desde una carpeta concreta.

Nos dirigiremos a la ruta donde está instalado MetaMap (por defecto su nombre será `public_mm`)

1. Creamos una carpeta, en caso de que no exista ya, llamada `sourceData`.

2. En el interior de esta carpeta, deberemos crear a su vez una carpeta nueva para el *workspace* con un nombre significativo. Desde MetaMap, recomiendan que el nombre tenga la forma:

   `<Año distribución>_<Algo siginificativo sobre distribución>`

   como por ejemplo: `12_custom`. A partir de ahora nos referiremos a él como `<nombre _workspace>`.

3. De nuevo a su vez, dentro de esta carpeta, se deberá generar otra nueva carpeta denominada `umls`.

4. Finalmente, dentro de la carpeta `umls`, se deberán copiar los archivos citados en el paso 7 del apartado anterior.

   Tras esto deberemos contar con la siguiente jerarquía de directorios: `public_mm/sourceData/<nombre_workspace>`.

Debido a que MetaMap y *DataFileBuilder* no han cambiado significativamente desde hace más de dos años, el sistema no se encuentra actualizado para poder usar la versión de 2012 de manera nativa, pero realizando lo siguiente se puede solventar:

1. Ir a la ruta: `public_mm/data/dfbuilder`
2. Ejecutar en el terminal la instruccion: `ln -s 2011 2012`.
3. Ir a la ruta: `public_mm/lexicon/data`
4. Ejecutar las siguientes instrucciones:

   ```
   ln -s lexiconStatic2012 lexiconStatic2011
   ln -s lexiconStatic2012IndByEui.dbx lexiconStatic2011IndByEui.dbx
   ln -s lexiconStatic2012IndByInfl.dbx lexiconStatic2011IndByInfl.dbx
   ```

C.4. Ejecución de *DataFileBuilder*

Una vez que se han generado los *enlaces simbólicos* del apartado anterior nos dirigimos a `public_mm` y ejecutamos el *script* `./bin/BuildDataFiles`.

Mientras este *script* está funcionando, se nos pedirán ciertas variables de configuración como serán:

- Nombre del *workspace*: en nuestro caso, como ya se ha mencionado, `12_custom`. Véase Creación del WorkSpace.
- Se nos preguntará si la ruta del *workspace* es correcta, si es correcta, simplemente pulsar la tecla *Enter*.
- Finalmente nos solicitan el año del que tomamos los datos, en nuestro caso: `2012`.

Esto nos generará en la carpeta de *workspace* las siguientes carpetas con diferentes archivos en su interior:

- `01metawordindex`
- `02treecodes`
- `03variants`
- `04synonyms`
- `05abbrAcronyms`

Así como el archivo `dfbuilder.profile`.

C.5. Ejecución de los diferentes *Scripts*

Una vez llegados a este punto, se ejecutarán los siguientes *scripts* en estricto orden, debido a que cada *script* suele tomar como datos los archivos generados por el *script* anterior. Para mayor comodidad, recomendamos situarnos en la carpeta `public_mm/sourceData/<nombre_workspace>`.

Nos dirigiremos a la carpeta `01metawordindex` y ejecutamos:

1. `./01/CreateWorkFiles`
2. `./02/Suppress`
3. `./03FilterPrep`
4. Ahora deberemos ejecutar uno de los 3 siguientes *scripts*. Será elegido en función de qué tipo de ontología queramos (*Strict*, *Moderate* o *Relaxed*).
   a) `./04FilterStrict`[4]
   b) `./04FilterModerate`
   c) `./04FilterRelaxed`
5. Una vez terminada la ejecución del *script* anterior (tardará, dependiendo de la máquina entre 14 horas y un día) se ejecutará el *script* `./05GenerateMWIFiles`.

Posteriormente deberemos situarnos en diferentes carpetas y ejecutar un determinado *script*.

En primer lugar nos dirigiremos a la carpeta `02treecodes` ejecutaremos `./01Generate Treecodes`.

Tras ello nos situamos en `03variants` y ejecutamos `./01GenerateVariants`. Este es el *script* que tiene un mayor tiempo de ejecución, conforme a lo indicado por él mismo, para un *Metashaurus* de 100000 entradas, tardará aproximadamente un día.

Seguidamente nos colocamos en la carpeta `04synonyms` y ejecutamos `./01Generate Synonyms`.

Finalmente nos dirigimos a `05abbrAcronyms` y ejecutamos el *script* `./01GenerateAbbr Acronyms`.

C.6. Trabajo Final

Cuando estamos en este punto, la gran mayoría de trabajo está realizado, únicamente nos queda ejecutar un *script* cuya duración aproximada es media hora.

1. Para ejecutarlo, volvemos a la carpeta `public_mm` y ejecutamos `./bin/LoadDataFiles`.

[4] Si se desea ejecutar este *script* (recomendado) deberá estar el `SKR/MedPOST` de MetaMap funcionando. Para ello se debe lanzar el *script*: `public_mm/bin/skrmedpostctl start`

2. Se nos solicitará el nombre del *workspace* y se nos pedirá que confirmemos si es correcta.
3. A continuación se requiere que introduzcamos el año de los datos, en nuestro caso `2012AA`.
4. Se nos preguntará si la ruta de destino es correcta (en caso de que así sea, pulsar *Enter*).
5. Finalmente se nos dirá si queremos cargar el modelo generado a lo que responderemos escribiendo `yes`.

Una vez acabado este *script*, tendremos en la carpeta de destino (confirmada el punto 4, y por defecto `public_mm/DB`) las siguientes carpetas:

- `DB.<nombre_workspace>.2012AA.base`
- `DB.<nombre_workspace>.2012AA.<modelo>`

C.7. Cambios para la utilización en *Windows*

Debido a que todo el proceso anterior se ha desarrollado para sistemas *UNIX*, las bases de datos generadas no se pueden utilizar en sistemas *Windows* sin realizar unos cambios minúsculos.

Los cambios son los siguientes:

1. Copiar y sustituir los siguientes archivos que están en `DB.<nombre_workspace>.2012AA.base` a la carpeta `DB.<nombre_workspace>.2012AA.<modelo>`:
   - `cuisourceinfo`
   - `cuisrc`
   - `meshmh`
   - `meshtcrelaxed`
   - `meshtcstrict`
   - `metamesh`
   - `metameshtc`
   - `nlsaa`
   - `nlsaau`
   - `sab_rv`
   - `sab_vr`
   - `syns`
   - `vars`
   - `varsan`
   - `varsanu`

- `varsu`

2. Copiar el archivo `config.02` en la carpeta `DB.<nombre_workspace>.2012AA.base` a la carpeta `DB.<nombre_workspace>.2012AA.<modelo>` y renombrarlo por `config`. Debemos tener en cuenta, que ese nombre de archivo ya existe, por lo que simplemente se elimina el archivo previo que existia y se renombra el archivo copiado.

Cabe destacar, que estos cambios no influyen en el funcionamiento en sistemas *UNIX* y lo único que nos permite es poder ejecutarlo en sistemas *Windows*.

Finalmente, para que MetaMapApi haga uso nuestro trabajo, se deberá ejecutar el servidor con los siguientes parámetros de configuración: `-V <nombre_workspace>`.

Apéndice D

Instalación y ejecución de MetaMap y MetaMap Java API

Este apéndice explica cómo realizar la instalación de MetaMap y MetaMap Java API y la manera en que deberá ejecutarse para que todo el sistema funcione correctamente.

D.1. Requisitos previos

Antes de poder poder realizar la instalación de MetaMap, deberemos descargarnos la ultima versión disponible, 2012 en nuestro caso.

1. El primer paso es registrarse en la UMLS para poder realizar la descarga, dicho registro puede llevarse a cabo desde:

   https://uts.nlm.nih.gov//license.html[1]

2. Una vez estemos registrados, nos dirigimos a la página que indicamos a continuación y seleccionamos la versión que queramos descargar en función de nuestro sistema operativo:

   http://metamap.nlm.nih.gov/#Downloads

3. A continuación, tenemos que descargar MetaMap Java API desde:

   http://metamap.nlm.nih.gov/#MetaMapJavaApi

4. Se recomienda guardar ambos archivos comprimidos en una misma carpeta que llamaremos de ahora en adelante `MetaMap`, que solo contendrá la informacion relativa a ello. Para evitar posibles problemas, siempre que sea posible evitaremos que la ruta en la que se encuentre esta carpeta no contenga espacios o símbolos de puntuación o acentuación.

5. Para poder realizar la instalación, además, deberemos contar con una versión de Java, y recomendamos que este instalado en la ruta por defecto. En caso de que no sea así, durante la instalación deberemos indicar la ruta adecuada cuando se le solicite.

[1] Dado que el registro debe ser aprobado por UMLS, puede no ser inmediato

D.2. Instalando MetaMap

Descomprimimos el archivo descargado en la carpeta que hemos llamado `MetaMap`.

La descompresión extraerá una carpeta llamada `public_mm`.

D.2.1. Instalación en un sistema Windows.

Únicamente deberemos abrir la carpeta `MetaMap/public_mm/` y ejecutar el `.bat`: `Install MetaMap.bat`.

Este script lanzará una interfaz grafica que nos guiara para la instalación de MetaMap. En ella se nos solicitarán la ruta en la que deseamos instalar MetaMap y la ruta en la que se encuentra la versión a utilizar de Java.

D.2.2. Instalación en un sistema GNU-Linux

Únicamente deberemos entrar la ruta `MetaMap/public_mm/bin/` y ejecutar `install.sh`.

D.3. Instalando MetaMap Java API

Descomprimimos el contenido del archivo descargado, de nuevo en la carpeta denominada `MetaMap`.

El contenido es a su vez una carpeta que se llama `public_mm`. Puesto que el nombre coincide con el del caso anterior, deberemos juntar la que ya teníamos con esta nueva, sustituyendo los archivos que entren en conflicto.

D.3.1. Instalación en un sistema Windows.

Para su instalación en un sistema Windows, deberemos entrar a la carpeta `MetaMap/-public_mm/bin` y ejecutar el script `install.bat`.

D.3.2. Instalación en un sistema GNU-Linux

Para la instalación en un sistema GNU-Linux, deberemos situarnos en la carpeta `MetaMap/-public_mm/bin` y ejecutar el *script* `install.sh` ya sea desde el terminal o haciendo doble clic en el archivo.

D.4. Ejecución MetaMap Java API

Ahora que ya tenemos descargados e instalados MetaMap y MetaMap Java API procederemos a ejecutar los servidores para que todo funcione correctamente.[2]

[2]Los servidores deben estar ejecutándose antes de que iniciar la aplicación.

D.4.1. Ejecución en un sistema Windows

En este caso, la instalación consiste en la ejecución de los tres archivos que se indican a continuación:

1. `MetaMap/public_mm/bin/skrmedpostctl_start.bat`
2. `MetaMap/public_mm/bin/wsdserverctl_start.bat`
3. `MetaMap/public_mm/bin/mmserver12`

   En nuestro caso, este último deberá lanzarse con las siguientes opciones:

   - `-y` (Para permitirnos el uso del *WSD*)
   - `-a` (Para permitirnos que estén todas las variaciones de acrónimos y abreviaturas)
   - `-V` (Para permitirnos la utilización de una versión personalizada de las bases de datos generada con el DataFileBuilder)

D.4.2. Ejecución en un sistema GNU-Linux

En este caso, la instalación consiste en la ejecución, desde el terminal, de los tres *scripts* que se indican a continuación:

1. `MetaMap/public_mm/bin/skrmedpostctl start`
2. `MetaMap/public_mm/bin/wsdserverctl start`

   (Al ejecutar este último perderemos el prompt).
3. `MetaMap/public_mm/bin/mmserver12`

   En nuestro caso, este último deberá lanzarse con las siguientes opciones:

   - `-y` (Para permitirnos el uso del *WSD*)
   - `-a` (Para permitirnos que estén todas las variaciones de acrónimos y abreviaturas)
   - `-V` (Para permitirnos la utilización de una versión personalizada de las bases de datos generada con el *DataFileBuilder*)
   - - UDA (Para permitirnos la utilización de un archivo `.txt` que se utilizará como apoyo en la detección de los acrónimos)

D.5. Creación del archivo UDA

MetaMap a partir de la versión del año 2011[3] permite la ejecución del sistema con un archivo llamado UDA (*User-Defined Acronyms and Abbreviations*) que, tal como su propio nombre indica contendrá aquellos acrónimos y abreviaturas definidos por el usuario.

La creación de este archivo es muy simple, y deberá tener el siguiente formato:

[3] http://metamap.nlm.nih.gov/MM_2011_ReleaseNotes.pdf

- `Acrónimo | Expansión`

También se acepta el caso contrario:

- `Expansión | Acrónimo`

De esta manera cuando MetaMap encuentre un acrónimo, lo cambiará por aquello que hemos indicado y realizando así la detección de conceptos con el acrónimo ya expandido.

D.6. Creación de los diferentes *Scripts*

Por comodidad, para evitar tener que repetir estos pasos cada vez que deseemos poner en ejecución MetaMap Java API, recomendamos realizar los siguientes *scripts*. Para que funcionen correctamente, estos deberán colocarse en la carpeta que en la que descomprimimos los archivos desde un prinicipio (y que decidimos en nuestro caso, denominar MetaMap), si se colocan en otra carpeta los *scripts* deberán modificarse convenientemente.

D.6.1. Scritps en Windows

Script1 Escribir en un archivo `.bat` el siguiente texto:

```
./public_mm/bin/skrmedpostctl_start.bat
```

Script2 Escribir en un archivo `.bat` el siguiente texto:

```
./public_mm/bin/wsdserverctl_start.bat
```

Scritp3 En este caso, contaremos con dos versiones del archivo. Cuando llegue el momento de ejecutar el *Script3* ejecutaremos una u otra versión en función de en qué idioma se encuentren los informes médicos que vayamos a utilizar, español o inglés.

Base_de_datos_en_Español Escribir en un archivo `.bat` el siguiente texto:

```
./public_mm/bin/mmserver12 -y -a -V 12_custom_esp
```

Base_de_datos_en_Ingles Escribir en un archivo `.bat` el siguiete texto:

```
./public_mm/bin/mmserver12 -y -a -V 12_custom_eng -UDA <Ruta del archivo
UDA>
```

Una vez contamos con estos archivos, lo único que debemos hacer es ejecutarlos en orden, recordando que para ejecutar el *Script3* deberemos escoger una única versión.

D.6.2. Scritp en GNU_Linux

Script1 Escribir en un archivo `.sh` el siguiente texto:

```
#! bin/bash
./bin/skrmedpostctl start
```

Script2 Escribir en un archivo `.sh` el siguiente texto:

```
#! bin/bash
./bin/wsdserverctl start
```

Scritp3 En este caso, contaremos con dos versiones del archivo. Cuando llegue el momento de ejecutar el *Script3* ejecutaremos una u otra versión en función de en qué idioma se encuentren los informes médicos que vayamos a utilizar, español o inglés.

Base_de_datos_en_Español Escribir en un archivo `.sh` el siguiente texto:

```
#! bin/bash
./bin/mmserver12 -y -a -V 12_custom_esp
```

Base_de_datos_en_Ingles Escribir en un archivo `.sh` el siguiente texto:

```
#! bin/bash
./bin/mmserver12 -y -a -V 12_custom_eng -UDA <Ruta del archivo UDA>
```

Una vez contamos con estos archivos, lo único que debemos hacer es ejecutarlos en orden, recordando que para ejecutar el *Script3* deberemos escoger una única versión.

Apéndice E

UCM at CLEF eHealth 2013

Las páginas de este apéndice contienen, ya que hemos creído oportuno incluirlo, tanto el *Extended Abstract* como el *paper* completo redactados como parte de nuestra participación en el *CLEF* 2013.

Abstract

Lucía Hervás Martín[1], Víctor Martínez Simón[2], Irene Sánchez Martínez[3] and Alberto Díaz Esteban[4]

University Complutense of Madrid, NIL, C/Profesor García Santesmases, Madrid, 28040, Spain

lhervasmartin@gmail.com, victormartinezsimon@gmail.com, irene.sanchzmartinz@gmail.com, albertodiaz@fdi.ucm.es

We are developing a system that analyze medical repords and extract a SNOMED-CT based concept representation. The more interesting characteristic of our system is not only that it can detect the concepts, it also take into account if they appear in an affirmative, negative or speculative context. The system also separates the concept representation according to the structure of the document, that is, there is a different representation for each section of the document.

Our system takes these steps: Automatic orthographic correction, Acronyms and abbreviation detector, Negation and speculation phrase detector, Concept detection.

For participating in Task 1 we have adapted our system in order to obtain the mentions that belong to the UMLS semantic group Disorders. The approach is based on using MetaMap to detect the concepts and the spans. Our aim was to identify what was the best way to use MetaMap in our system to solve the Task 1.

To detect and analyse the different concepts into the medical report, we use Metamap and the SKR/MedPost server included into the Metamap distribution. The two of them in the lastest version (2012).

We send two different results, the 2012AA USAbase strict model for the first run and the 2011AA USAbase strict model for the second run.

Our best results for task 1a shown a 0.504 F1 score with strict evaluation, and a 0.660 F1 score with relaxed evaluation.

Our best results for task 1b shown a 0.362 Acurracy with strict evaluation and a 0.870 Accuracy with relaxed evaluation.

UCM at CLEF eHealth 2013 Shared Task1

Lucía Hervás Martín[1], Víctor Martínez Simón[2], Irene Sánchez Martínez[3], and Alberto Díaz Esteban[4]

University Complutense of Madrid, NIL, C/Profesor García Santesmases, Madrid, 28040, Spain,
lhervasmartin@gmail.com
victormartinezsimon@gmail.com
irene.sanchzmartinz@gmail.com
albertodiaz@fdi.ucm.es,
http://www.ucm.es/

Abstract. We are developing a system that analyze medical reports and extract a SNOMED-CT based concept representation. The more interesting characteristic of our system is not only that it can detect the concepts, it also take into account if they appear in an affirmative, negative or speculative context. The system also separates the concept representation according to the structure of the document.
Our system takes these steps: automatic orthographic correction, acronyms and abbreviation detection, negation and speculation phrase detection and medical concepts detection.
For participating in Task 1 we have adapted our system in order to obtain the mentions that belong to the UMLS semantic group Disorders. The approach is based on using MetaMap to detect the concepts and the spans. Our aim was to identify what was the best way to use MetaMap in our system to solve the Task 1.

Keywords: Natural Language Processing, medical report, spellcheck, acronym expansion, negated phrase, speculated phrase, concept detection, Metamap, UMLS

1 Introduction

With the advent of computers and especially the Internet, the information explosion produced has grown by leaps and bounds.We are in a time where progress allow us to generate, store and distribute lots of information. But all this advances have disadvantages too.

By this moment lots of information are generated, and differentiate the correct information from the wrong information requires a big amount of money and time.

With the aim of reduce this two problems, the computer engineers, the software developers and the medicine experts, are developing different strategies to extract information from different places and increase the medical knowledge, to improve researches and in the last term, to have a better live.

With this aim in mind, we decide to develop a tool to help medical experts in their job.

During our research, we discover the CLEF task for biomedical documents which is very close of what we are developing so we decide to participate an increase our knowing and the performance of our system.

We develop software to solve the first task that consist in discover where are the different medical concepts in a medical report and catalogue them, using for this the UMLS CUIs.

We are developing a system that analyze medical records and extract a SNOMED-CT based concept representation. Before the analysis we have other phases: a language corrector and an acronyms expander. The more interesting characteristic of our system is that not only detect the concepts, it also take into account if they appear in an affirmative, negative or speculative context. The system also separates the concept representation according to the structure of the document, that is, there is a different representation for each section of the document.

For participating in Task 1 we have adapted our system in order to obtain the mentions that belong to the UMLS semantic group Disorders. The approach is based on using MetaMap to detect the concepts and the spans. Our aim was to identify what was the best way to use MetaMap in our system to solve the Task1.

We have submitted runs with no external annotations, two for task 1a and two for task 1b. The difference between the runs is only the DB used. We used the 2012AA USAbase strict model for the first run and the 2011AA USAbase strict model for the second run. Our best results for task 1a shown a 0.504 F1 score with strict evaluation, and a 0.660 F1 score with relaxed evaluation. Our best results for task 1b shown a 0.362 Accuracy with strict evaluation and a 0.870 Accuracy with relaxed evaluation.

2 State of the art

The aim of this part of the text it's explain the different tools and approximations that actually exists in the world of natural language processing and which of them we use in our project and are used in the software developed for Clef task.

The different tools analyzed were:

- Ontologies
- MetaMap
- Concept detection and disambiguation
- Orthographic correction
- Negation detection
- Speculation detection

2.1 Ontologies

One definition of what it is an ontology is this: "An ontology defines the basic terms and relations comprising the vocabulary of a subject area, as well as the

rules for combining these terms and relations define extensions to the vocabulary"[7].

In medical and biomedical reports, there are two important ontologies: SNOMED-CT and UMLS

- SNOMED-CT:
  SNOMED-CT or Systematized Nomenclature of Medicine Clinical Terms is a big ontology of medical concepts, written in different idioms and by this moment it's used in more than fifty different countries. Since 2007, the SNOMED-CT license belongs to the International Health Terminology Standards Development Organization (IHTSDO) which distributed and maintains the Ontology.
  SNOMED-CT it's compound by this elements:
  - Concepts: They represent different medical terms. All of them, have a unique identifier.
  - Descriptors: A concept could have more than one different descriptor, which are synonyms between them.
  - Relationships: These are use to connect different concepts because they have any kind of relationship.
- UMLS:
  UMLS or Unified Medical Language System is a file and software set which have a big amount of biomedical terms. It was developed by the National Library of Medicine (NLM). It's compound by 4 kind of elements.
  - Metathesaurus:
    It's a big database with lots of biomedical concepts. It takes as sources some databases, for example: MeSH, RxNorm and SNOMED-CT.
    All elements in the Metathesaurus gets an identification number to represent them.
  - Semantic type:
    The semantic type it's used to classify the biomedical vocabulary and it also contains the different relationships between all the concepts.
  - Specialized lexicon.
    A specialized lexicon its incorporated into MetaMap distribution, which have more than 200.000 different terms and common English words. This terms are use to improve the different lexical tools included in the distribution.
  - Different lexical tools like:
    * A Lexical Variant Generator (LVG).
    * A word index generator (WordInd).
    * A normalized strings generator (Norm)

2.2 MetaMap

The MetaMap program has been used extensively for a wide array of BioNLP studies, such as automatic indexing of biomedical literature and concept based text summarization.

MetaMap finds Metathesaurus concepts by performing a shallow syntactic analysis of the input text, producing a set of noun phrases. The noun phrases are then used to generate sets of variants which are consequently looked up from the Metathesaurus concepts. Matching concepts are evaluated against the original text and the strength of the mappings are calculated. The candidates are finally combined and the final scores are computed, where the highest score of a complete mapping represents MetaMap's interpretation of the text.

The MetaMap program is provided to be run both locally and remotely. We ran the current version of MetaMap, MetaMap2012 remotely via the batch mode facility.

The parameter set that influences the performance of MetaMap included; using a Relaxed model, selecting the NLM2102AB Metathesaurus version, including all derivational variants, enabling unique acronym/abbreviation variants only, allowing large N, preferring multiple concepts and using word sense disambiguation.

Relaxed Model is a model provided by MetaMap in addition to the Strict model. MetaMap offers the Strict model as a default setting in which all types of filterings are applied, however, we applied the Relaxed model in which only Manual and Lexical filterings are used. While the Strict model is most appropriate for experiments that require the highest accuracy, it only covers only 53% of the Metathesaurus strings. As we consider high coverage of concepts an important factor, we thus applied the relaxed model which consists of up to 83% of Metathesaurus strings.

The versions of Metathesaurus, Base, USAbase and NLM, provided with MetaMap are different in their Metathesaurus coverage and the license type required for using vocabulary sources. The NLM version which is offered at no cost for research purposes and covers all of the provided Metathesaurus was used in our work.

Variants, such as inflectional and derivational variants, are computed by MetaMap to account for the textual variation in the text. With this setting, many types of variants are generated recursively, and only acronyms and abbreviations are restricted to the unique ones. In addition, the candidates also include words that can be prepositions, conjunctions or determiners if they occur often enough in Metathesaurus.

Prefer multiple concepts causes MetaMap to score the mappings with more concepts higher than those with fewer concepts. This option is useful for discovering higher-order relationships among concepts found in the text and as such is assumed to be helpful.

Word sense disambiguation attempts to solve lexical ambiguities by identifying the correct meaning of a word based on its context. By using this option in MetaMap, the program attempts to solve the ambiguities among equally scoring concepts by choosing the concept(s) based on semantic type.

2.3 Concept detection and disambiguation.

When we are detecting a concept, we can't only search any word in the text into a database, because is high probably that this same concept has more than one entrance into the database. To do our work, we start searching different ways to detect and disambiguate concept, and we decide to work with MetaMap software.

As we have just said, MetaMap, includes concept disambiguation, with a very good result ratio.

They use an algorithm written on C++, that after a few time, it was written into Java and Perl. This algorithm is based in the Hidden Markov Model and was modify to include contextual and grammatical information to improve the system accuracy.

The developers also using a training set, learn which words can go with another words based on their grammatical category [11].

With all these improvements, they made an algorithm that with a 1000 different phrases, they have a 97,43% precision with 582 phrases correctly marked and 261 phrases with only one error.

2.4 Orthographic correction.

When a doctor it's writing a medical report, he or she may have a misspelling because he or she is writing very fast and taking notes. A little orthographic error can change the meaning of all the phrase so good written reports are recommended.

We can say that an orthographic corrector it's an algorithm that checks every word into a database and says if this word is correct or if it's incorrect and in this case, says some corrections ordered by a score. The best of all these algorithm is the one who says the better suggestions. To increase the suggestions ratio, they use some different methods like:

- Distance between words
  With this kind of improvement, we check how far are some words from another words and the closest words are probably the correct one. There are some different ways to develop this issue. Here are some of them:
  - Differences between characters
  - Hamming distance [4]
  - Levenshtein distance [6]
  - Damerau-Levenshtein distance [2]
  - Keyboard distance as Manhattan distance using Needleman-Wunsch algorithm [5] [8]
- Phonetic algorithms
  All of this improvements check how close are different phonetic sounds between all the phonetic sounds from the two words. The different algorithms for this issue are:
  - Soundex [10]
  - New York State Identification and Intelligence System (NYSIIIS) [12]
  - Metaphone [9]

2.5 Negation detection.

When we talk and write, we don't say every phrases affirmed, we use negative phrase to emphasize different things. On medical reports, negated phrases are very important to detect, because they say some concepts that the patient doesn't have and if we misinterpret this phrases and concepts, we will go in the wrong direction.

We use 2 approximations to solve this problem. The first was use the NegEx algorithm as is distributed, and the second option was use the MetaMap NegEx algorithm implementation.

NegEx algorithm The NegEx algorithm [1] uses regular expressions to detect when a phrase is negated and say witch word of all causes this detection.

The algorithm uses as regular expressions, four kind of concepts:

- Negative words: these are the words that negate all the phrase that appear right after them.
- PostNegative words: these are the words that negate the phrase right before them.
- Conjunctions: these are the words that connect different negative phrases.
- Pseudo Negations: these are the words that we may think that make a phrase negative, but the really don't do that, so when we detect this words, we skip them.

NegEx on MetaMap MetaMap include the NegEx algorithm, but in the year 2009, the MetaMap developers improve the algorithm[1].

The improvements where this:

- Add new words to detect negations like: *other than, otherwise, to account for, to explain* and *then.*
- Add to MetaMap databases, some negated concepts. If any of this concepts appear in the text, then the phrase is negated.
- Join different negated concepts into one, to reduce noise.

2.6 Speculation detection.

The speculation detection it's a very important task in biomedical report. Sometimes, specialist aren't really sure about what are they detecting, because some different illness have the same symptoms.

To resolve this issue, after a web search, we learn how to adapt the NegEx algorithm to detect the speculation phrases into a medical report [3].

[1] http://metamap.nlm.nih.gov/MM09_Release_Notes.shtml

3 Framework evaluation

Participants will be provided training and test datasets. The evaluation for all tasks will be conducted using the withheld test data. Participating teams are asked to stop development as soon as they download the test data. Teams are allowed to use any outside resources in their algorithms. However, system output for systems that use annotations outside of those provided for Tasks 1 will be evaluated separately from system output generated without additional annotations.

3.1 Evaluation Measures

Task 1 - Named entity recognition and normalization of disorders

A. Boundary detection of disorders: identify the span of all named entities that could be classified by the UMLS semantic group Disorder (excluding the semantic type Findings)

Evaluation measure: F1-score

- F1-score = (2 * Recall * Precision) / (Recall + Precision)
- Recall = TP / (TP + FN)
- Precision = TP / (TP + FP)
- TP = same span
- FP = spurious span
- FN = missing span

Exact F1-score: span is identical to the reference standard span

Overlapping F1-score: span overlaps reference standard span

B. Named entity recognition and normalization of disorders: identify the boundaries of disorders and map them to a SNOMED-CT code.

Evaluation measure: Accuracy

- Accuracy = Correct/Total
- Correct = Number of disorder named entities with strictly correct span and correctly generated code
- Total = Number of disorder named entities, depending on strict or relaxed setting:
  - Strict: Total = Total number of reference standard named entities. In this case, the system is penalized for incorrect code assignment for annotations that were not detected by the system.
  - Relaxed: Total = Total number of named entities with strictly correct span generated by the system. In this case, the system is only evaluated on annotations that were detected by the system.

4 Processing

As we have just say, we are developing another system that extract affirmative, negative and speculative concepts from medical reports and when we decide participate on Clef tasks, we made a few changes in our software to satisfy the requisites. Here we will explain what our system does and which changes we made to adjust to Clef tasks.

4.1 The software tool

The first action that is done in our system it's transform the medical report to a XML file, so we can use it easily. To do this little task, we use different and configurable regular expressions, where we define different "Beginning words" witch means the beginning of any different sections in our report. For example, we have this sections: "Allergies", "Family history". When a "Beginning word" is detected, we assume that in this moment, a new sections begins and it continues until the end of the report or another "Beginning Word".

After that, we made a orthographic correction using the Hunspell's algorithms and dictionaries. We only made a correction if there is a lot of score in the suggestion, because if we correct all the things that Hunspell suggest, we'll probably introduce some noise that in future actions will be very dangerous.

The next step that we do, it's detect acronyms and abbreviation, using for that 3 kinds of detectors:

1. A Database that we have developed, witch contains some different acronyms and abbreviation and a few rules to make a little disambiguation.
2. The MetaMap acronym and abbreviation detector.
3. Using MetaMap with an UDA file.

The next step in our system it's the negation detection. To do this issue first we think in the NegEx algorithm but MetaMap includes a modified NegEx algorithm so we use this last option.

After this last step, we have a speculation detector which works very easily. We have a NegEx algorithm modified to detect some speculative words as regular expressions.

Finally, we have the concept detector.

To do this finally task, we use MetaMap to detect the different concepts and to know their CUI. MetaMap retrieves some concepts, so to reduce the noise, we only take the concepts with the most score of all of them. We also use the MedPost/SKR server included in MetaMap to have a disambiguation between concepts. Another noise reductive technique that we develop, was a custom dataset of concepts using to create it the MetamorphoSys tool and the DataFileBuilder developed by the MetaMap group.

After all this job, we save the results in a XML file to use them in the future for different things, like use Lucene to search between different medical reports.

4.2 Modifications for CLEF task

We made a several modifications to make our system complete the Clef tasks.

First we disable the language corrector, because we assume that all the reports will be right written.

We also disable the acronym and abbreviation detector because this is the task 2 target.

We disable the negation and speculative phrase detector because this will be useful in task 3, but to detect concepts, it only make the system run slowly.

Finally, we make the system to accept only the next semantic types:

- Congenital Abnormality
- Acquired Abnormality
- Injury or Poisoning
- Pathologic Function
- Disease or Syndrome
- Mental or Behavioral Dysfunction
- Cell or Molecular Dysfunction
- Experimental Model of Disease
- Anatomical Abnormality
- Neoplastic Process
- Signs and Symptoms

5 Results

For participating in Task 1 we have adapted our system in order to obtain the mentions that belong to the UMLS semantic group Disorders. The approach is based on using MetaMap to detect the concepts and the spans. We have submitted runs with no external annotations, two for task 1a and two for task 1b. The difference between the runs is only the DB used. We used the 2012AA USAbase strict model for the first run and the 2011AA USAbase strict model for the second run. Our best results for task 1a shown a 0.504 F1 score with strict evaluation, and a 0.660 F1 score with relaxed evaluation. In strict evaluation 18th and 20th of 27 runs. In relaxed evaluation, 21th and 22th of 28 runs. Explain the differences between strict and relaxed evaluation. Explain the differences between using 2012AA or 2011AA.

Table 1. Task 1A. No external annotations. Strict

Team,Country	Precision	Recall	F1-Score
UTHealth_CCB.2, UT, USA	0.800	0.076	0.750
NIL-UCM.2, Spain	0.617	0.4626	0.504
NIL-UCM.1, Spain	0.621	0.4616	0.498
FAYOLA.1, VW, USA	0.024	0.446	0.046

Table 2. Task 1A. No external annotations. Relaxed

Team,Country	Precision	Recall	F1-Score
UTHealth_CCB.2, UT, USA	0.925	0.827	0.873
NIL-UCM.2, Spain	0.809	0.558	0.660
NIL-UCM.1, Spain	0.812	0.543	0.651
FAYOLA.1, VW, USA	0.504	0.043	0.079

Table 3. Task 1B. No external annotations. Strict

Team,Country	Accuracy(sn2012)	Accuracy(sn2011)
NCBI.2, MD, USA	0.589	0.584
NIL-UCM.2, Spain	0.362	0.362
NIL-UCM.1, Spain	0.362	0.362
NCBI.2, MD, USA	0.006	0.006

Table 4. Task 1B. No external annotations. Relaxed

Team,Country	Accuracy(sn2012)	Accuracy(sn2011)
AEHRC.1, QLD, Australia	0.939	0.939
NIL-UCM.1, Spain	0.871	0.870
NIL-UCM.2, Spain	0.850	0.850
UTHealth_CCB.1, UT, USA	0.728	0.772

6 Errors and Analysis

With respect to Task 1a, our system have a lot of false positive, because the only word sense disambiguation that we use, it's with the MedPost/SKR server included in MetaMap distribution, and better with the databases of 2011.

With respect to Task 1b, we have a very good ratio detection of CUIs. MetaMap returns a lot of possible candidates, but we decide to save only the one with bigger score and in case of there are more than one with the same score, the first to appear in the list. We fail in this task for two reasons:

1. We don't take a look on the other results that MetaMap retrieve, so maybe the good one it's the second with most score.
2. We have the same error than in task 1a, we don't have a good word sense disambiguation, so we have lots of false positive detections.

References

1. Chapman, W. W., Bridewell, W., Hanbury, P., Cooper, G. F., and Buchanan, B. G.: A simple algorithm for identifying negated findings and diseases in discharge summaries. Journal of biomedical informatics, vol. 34, no. 5, pp. 301-310, (2001)
2. Damerau, F. J.: A technique for computer detection and correction of spelling errors. Communications of the ACM, vol. 7, no. 3, pp. 171-176, (1964)
3. Farkas, R., Vincze, V., Mra, G., Csirik, J., and Szarvas, G.: The CoNLL-2010 shared task: learning to detect hedges and their scope in natural language text. in Proceedings of the Fourteenth Conference on Computational Natural Language Learning-Shared Task. Association for Computational Linguistics, (2010), pp. 1-12
4. Hamming, R. W.: Error detecting and error correcting codes. Bell System technical journal, vol. 29, no. 2, pp. 147-160, (1950)
5. Krause, E. F.: Taxicab geometry: An adventure in non-Euclidean geometry. Courier Dover Publications, (1987)
6. Levenshtein, V. I.: Binary codes capable of correcting deletions, insertions and reversals. in Soviet physics doklady, vol. 10, (1966), p. 707
7. Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W.: Enabling technology for knowledge sharing. AI magazine, vol. 12, no. 3, p. 36, (1991)
8. Needleman, S. B. and Wunsch, C. D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology, vol. 48, no. 3, pp. 443-453, (1970)
9. Philips, L.: Hanging on the metaphone. Computer Language, vol. 7, no. 12 (December), (1990)
10. Russell, R. and Odell, M.: Soundex. US Patent, vol. 1, (1918)
11. Smith, L., Rindflesch, T., and Wilbur, W. J.: MedPost: a part of speech tagger for biomedical text. Bioinformatics, (2004)
12. Taft, R. L.: Name search techniques. Bureau of Systems Development, New York State Identification and Intelligence System, (1970), no. 1

Apéndice F

Glosario de términos

Aspell: corrector ortográfico de software libre, es el corrector estándar para los sistemas de software *GNU*.

CLEF: *Conference and Labs of the Evaluation Forum* es una organización cuyo principal objetivo es la investigación, innovación, y búsqueda de la información en sistemas multi-lenguaje y con varios niveles de estructuración.

Hunspell: corrector ortográfico y analizador morfológico diseñado para idiomas con una morfología rica, compleja formación de palabras compuestas o con una codificación de caracteres distinta del ASCII de 8 bits.

Lexicón: diccionario.

Lucene: API de código abierto para recuperación de información, especialmente útil para cualquier aplicación que requiera indexado y búsqueda a texto completo.

MetaMap: software desarrollado por el Dr. Alan Aronson en la *NLM* de Estados Unidos que cuenta con múltiples opciones de configuración y permite mapear o asociar términos que aparecen en un texto biomédico, con conceptos del Metatesauro *UMLS*.

Metatesauro: tesauro multi-lenguaje que contiene millones de conceptos biomédicos y sanitarios, sus nombres, sinónimos y sus relaciones utilizado en las diferentes herramientas de *UMLS*.

MXML: extensión para ficheros `.xml`. Es un formato propio de nuestra aplicación que permite identificar a los informes mapeados.

Ontología: formulación de un exhaustivo y riguroso esquema conceptual dentro de uno o varios dominios dados, con la finalidad de facilitar la comunicación y el intercambio de información entre diferentes sistemas y entidades.

PLN: el Procesamiento del Lenguaje Natural consiste en un conjunto de técnicas de la rama de la inteligencia artificial cuyo objetivo es conseguir una comunicación hombre-máquina usando para ello el lenguaje con el que se comunican los hombres entre sí.

Prolog: lenguaje de programación mediante el paradigma lógico con técnicas de producción final interpretada.

PXML: extensión para ficheros `.xml`. Es un formato propio de nuestra aplicación que permite identificar a los informes procesados.

Sistema experto: aplicación informática capaz de solucionar un conjunto de problemas que exigen un gran conocimiento sobre un determinado tema.

***SNOMED-CT*:** *Systematized Nomenclature of Medicine Clinical Terms* es la terminología clínica integral, multi-lenguaje y codificada de mayor amplitud, precisión e importancia desarrollada en el mundo.

SQLite: sistema de gestión de bases de datos relacional.

Tesauro: lista de palabras con significados similares, sinónimos, habitualmente acompañada por otra lista de antónimos. Normalmente está reducido a un campo concreto de la lengua como puede ser el ámbito biomédico

***UMLS*:** *Unified Medical Language System* consite conjunto de archivos y software que reúne gran cantidad de vocabularios biomédicos y de salud, y los estándares que dan la posibilidad de que distintos sistemas informáticos operen entre sí

Bibliografía

[1] A. R. Aronson, "Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program." in *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 2001, p. 17. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2243666/

[2] A. R. Aronson, "Metamap: Mapping text to the umls metathesaurus," *Bethesda, MD: NLM, NIH, DHHS*, 2006. [Online]. Available: https://lib.njnu.edu.cn/proxy/Enlinkwkkoxe00ltm1qsr1qvw1xxpi/papers/references/metamap06.pdf

[3] A. R. Aronson and F.-M. Lang, "An overview of MetaMap: historical perspective and recent advances," *Journal of the American Medical Informatics Association*, vol. 17, no. 3, pp. 229–236, 2010. [Online]. Available: http://jamia.bmjjournals.com/content/17/3/229.abstract

[4] K. Atkinson, "GNU Aspell Spell Checker," 2004.

[5] E. B. Barahona, I. S. Ramos, and A. U. na Herradón, "Procesador automático de informes médicos," 2012, proyecto de Sistemas Informáticos (Facultad de Informática, Curso 2011-2012). [Online]. Available: http://eprints.ucm.es/16070/

[6] O. Bodenreider, "Biomedical Ontologies: Session Introduction O. Bodenreider, JA Mitchell, and AT McCray Pacific Symposium on Biocomputing 8: 562-564 (2003)," in *Pacific Symposium on Biocomputing*, vol. 8, 2003, pp. 562–564. [Online]. Available: http://psb.stanford.edu/psb-online/proceedings/psb03/intro-ont.pdf

[7] M. Cabré, "La terminología hoy: concepciones, tendencias y aplicaciones," *Ciência da Informação*, vol. 24, no. 3, 1995.

[8] W. W. Chapman, W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan, "A simple algorithm for identifying negated findings and diseases in discharge summaries," *Journal of biomedical informatics*, vol. 34, no. 5, pp. 301–310, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1532046401910299

[9] C. W. Cleverdon, "On the inverse relationship of recall and precision," *Journal of Documentation*, vol. 28, no. 3, pp. 195–201, 1972.

[10] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[11] A. Díaz, L. Plaza, V. Francisco, P. Gervás, E. Mota, A. Romero, I. Colodrón, A. Palacios, and O. Partida, "AutoIndexer: Investigación y Desarrollo de Metodologías y Recursos Terminológicos de Apoyo para los Procesos de Indexación Automática de Documentos Clínicos," *Procesamiento de Lenguaje Natural*, vol. 47, pp. 343–344, 2011.

[12] R. Farkas, V. Vincze, G. Móra, J. Csirik, and G. Szarvas, "The CoNLL-2010 shared task: learning to detect hedges and their scope in natural language text," in *Proceedings of the Fourteenth Conference on Computational Natural Language Learning—Shared Task*. Association for Computational Linguistics, 2010, pp. 1–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870536

[13] T. e. a. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993. [Online]. Available: http://secs.ceas.uc.edu/~mazlack/ECE.716.Sp2011/Semantic.Web.Ontology.Papers/Gruber.93a.pdf

[14] N. Guarino, *Formal ontology in information systems: proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. Ios PressInc, 1998, vol. 46.

[15] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[16] E. Hatcher, O. Gospodnetic, and M. McCandless, "Lucene in action," 2004.

[17] D. Hood, "Caverphone: Phonetic matching algorithm," *Technical Paper CTP060902, University of Otago, New Zealand*, 2002.

[18] S. Jacquemont, F. Jacquenet, and M. Sebban, "Correct your text with Google," in *Web Intelligence, IEEE/WIC/ACM International Conference on*, 2007, pp. 170–176.

[19] E. F. Krause, *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Dover Publications, 1987.

[20] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in *Soviet physics doklady*, vol. 10, 1966, p. 707.

[21] MySQL, "5.5 reference manual," 2011.

[22] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout, "Enabling technology for knowledge sharing," *AI magazine*, vol. 12, no. 3, p. 36, 1991. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/902

[23] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[24] Oracle, "Oracle Database SQL Language Reference," 2012. [Online]. Available: http://docs.oracle.com/cd/E11882_01/server.112/e17118/functions167.htm

[25] L. Philips, "Hanging on the metaphone," *Computer Language*, vol. 7, no. 12 (December), 1990.

[26] L. Philips, "The double metaphone search algorithm," *C/C++ users journal*, vol. 18, no. 6, pp. 38–43, 2000.

[27] PHP, "PHP Manual," 2013. [Online]. Available: http://php.net/manual/en/function.metaphone.php

[28] R. Russell and M. Odell, "Soundex," *US Patent*, vol. 1, 1918.

[29] A. S. Schwartz and M. A. Hearst, "A simple algorithm for identifying abbreviation definitions in biomedical text," in *Pacific Symposium on Biocomputing 2003: Kauai, Hawaii, 3-7 January 2003*. World Scientific Publishing Company Incorporated, 2002, p. 451.

[30] P. H. Sellers, "On the theory and computation of evolutionary distances," *SIAM Journal on Applied Mathematics*, vol. 26, no. 4, pp. 787–793, 1974.

[31] L. Smith, T. Rindflesch, and W. J. Wilbur, "MedPost: a part of speech tagger for biomedical text," *Bioinformatics*, 2004. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/early/2004/04/08/bioinformatics.bth227.short

[32] R. L. Taft, *Name search techniques*. Bureau of Systems Development, New York State Identification and Intelligence System, 1970, no. 1.

[33] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[34] Wikipedia, "Hidden Markov model — Wikipedia, The Free Encyclopedia," 2013, [Online; accessed 3-June-2013]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Hidden_Markov_model&oldid=558841924

[35] Wikipedia, "Lucene — Wikipedia, The Free Encyclopedia," 2013, [Online; accessed 2-June-2013]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Lucene&oldid=554589509

[36] Wikipedia, "Sequence alignment — Wikipedia, The Free Encyclopedia," 2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Sequence_alignment&oldid=559387915